TUGAS AKHIR – KI141502

RANCANG BANGUN PLUG-IN ECLIPSE UNTUK MEREKAM AKTIVITAS PEMROGRAMAN MENGGUNAKAN FINITE STATE MACHINE

ANDRE ZACHARY REINALDI
NRP 5113100144

Dosen Pembimbing I
RIZKY JANUAR AKBAR, S.Kom., M.Eng.

Dosen Pembimbing II
Ir. F.X. ARUNANTO, M.Sc.

DEPARTEMEN TEKNIK INFORMATIKA
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya 2017

TUGAS AKHIR – KI141502

RANCANG BANGUN PLUG-IN ECLIPSE UNTUK MEREKAM AKTIVITAS PEMROGRAMAN MENGGUNAKAN FINITE STATE MACHINE

ANDRE ZACHARY REINALDI
NRP 5113100144

Dosen Pembimbing I
RIZKY JANUAR AKBAR, S.Kom., M.Eng.

Dosen Pembimbing II
Ir. F.X. ARUNANTO, M.Sc.

DEPARTEMEN TEKNIK INFORMATIKA
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya 2017

UNDERGRADUATE THESES – KI141502

ECLIPSE PLUGIN DESIGN TO CAPTURE CODING ACTIVITY USING FINITE STATE MACHINE

ANDRE ZACHARY REINALDI
NRP 5113100144

First Advisor
RIZKY JANUAR AKBAR, S.Kom., M.Eng.

Second Advisor
Ir. F.X. ARUNANTO, M.Sc.

Department of Informatics
Faculty of Information Technology
Sepuluh Nopember Institute of Technology
Surabaya 2017

(Halaman ini sengaja dikosongkan)

LEMBAR PENGESAHAN

RANCANG BANGUN PLUG-IN ECLIPSE UNTUK MEREKAM AKTIVITAS PEMROGRAMAN MENGGUNAKAN FINITE STATE MACHINE

TUGAS AKHIR

Diajukan Untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Bidang Studi Algoritma Pemrograman
Program Studi S-1 Departemen Teknik Informatika
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember

Oleh:
ANDRE ZACHARY REINALDI
NRP: 5113100144

Disetujui oleh Pembimbing Tugas Akhir:

1. **Rizky Januar Akbar, S.Kom, M.Eng.** ..... (Pembimbing 1)
   (NIP. 198701032014041001)

2. **Ir F.X. Arunanto, M.Sc.** ..... (Pembimbing 2)
   (NIP. 195701011983031004)

SURABAYA
JUNI, 2017

(Halaman ini sengaja dikosongkan)

RANCANG BANGUN *PLUG-IN* ECLIPSE UNTUK MEREKAM AKTIVITAS PEMROGRAMAN MENGGUNAKAN *FINITE STATE MACHINE*

Nama Mahasiswa : ANDRE ZACHARY REINALDI
NRP : 5113100144
Jurusan : Teknik Informatika FTIF-ITS
Dosen Pembimbing 1 : Rizky Januar Akbar, S.Kom, M.Eng.
Dosen Pembimbing 2 : Ir F.X. Arunanto, M.Sc.

Abstrak

Dokumentasi mengenai *source code* program yang ada saat ini sangat berbagai macam bentuknya. Namun seringkali dalam dokumentasi tersebut urutan proses pemrograman hilang, padahal jika urutan itu bisa didapatkan maka banyak sekali manfaat yang diperoleh. Contoh dokumentasi yang menyimpan perubahan dalam *code* adalah VCS. Namun yang dapat terlihat dalam VCS adalah perbedaan antara dokumen versi terbaru dengan versi sebelumnya, tidak terdapat urutan terjadinya perubahan dalam *code* tersebut. Dalam sebuah penelitian sebelumnya telah dibuat sebuah *plug-in* yang dapat mendeteksi jumlah perubahan yang terjadi dalam *source code*. Penulis menawarkan sebuah *plug-in* Eclipse untuk merekam aktivitas pemrograman secara langsung saat *user* menuliskan kode dengan menggunakan konsep *finite state machine*. *Plug-in* dapat mencatat urutan aktivitas pemrograman yang dibuat oleh *programmer*. *Plug-in* yang dibuat berhasil mencatat dan mengklasifikasikan aktivitas pemrograman secara berurutan yang dituliskan oleh *user* namun aktivitas yang dituliskan tidak terkait dengan konteks.

Kata kunci: Code, Keystroke, Programming Activity, Syntax, Finite state machine, Document Listener

(Halaman ini sengaja dikosongkan)

ECLIPSE PLUGIN DESIGN TO CAPTURE CODING ACTIVITY USING FINITE STATE MACHINE

Student's Name	**: ANDRE ZACHARY REINALDI**
Student's ID	**: 5113100144**
Department	**: Department of Informatics FTIF-ITS**
First Advisor	**: Rizky Januar Akbar, S.Kom, M.Eng.**
Second Advisor	**: Ir F.X. Arunanto, M.Sc.**

Abstract

Today, source code documentation is very diverse. But those documentation forms do not contain the sequence of programming. As a matter of facts, if that sequence can be obtained then there are many benefits that can be taken from it. There is documentation form that holds the record of the revision in the document such as Github or bitbucket, but it only holds a version of code only when the code was saved in the system. In an earlier research, a plug-in was built to record a change that happened in the source code. The writer suggests an Eclipse plug-in to capture and record programming activity live when the code is written by the programmer using a finite state machine concept. Plug-in can record the sequence of written code. A Plug-in that has been created, performed well to record and classify programming activity sequentially that was written by the user but the classified activity is still context free.

Keywords: Code, Keystroke, Programming Activity, Syntax, Finite state machine, Document Listener

(Halaman ini sengaja dikosongkan)

KATA PENGANTAR

بِسۡمِ ٱللَّهِ ٱلرَّحۡمَٰنِ ٱلرَّحِيمِ

Alhamdulillahirabbil'alamin, segala puji bagi Allah Swt yang telah melimpahkan rahmat dan hidayah-Nya sehingga penulis dapat menyelesaikan Tugas Akhir yang berjudul:

"**RANCANG BANGUN PLUG-IN ECLIPSE UNTUK MEREKAM AKTIVITAS PEMROGRAMAN MENGGUNAKAN FINITE STATE MACHINE**"

yang merupakan salah satu syarat dalam menempuh ujian sidang guna memperoleh gelar Sarjana Komputer. Selesainya Tugas Akhir ini tidak terlepas dari bantuan dan dukungan beberapa pihak, sehingga pada kesempatan ini penulis mengucapkan terima kasih kepada:

1. Allah SWT karena atas rahmat dan limpahannya penulis dapat menyelesaikan studi dari awal sampai ke tugas akhir di Departemen Teknik Informatika ITS.
2. Ibu Khomsatun dan Bapak Rubianto yang telah membimbing, memberikan semangat dalam belajar. Dan juga tidak lelah dalam mengingatkan untuk selalu belajar dan sholat.
3. Antoni Ridzki Andromeda yang selalu memberikan bantuan baik secara moral maupun financial dimanapun dan kapanpun penulis membutuhkan saran dan nasihat.
4. Bapak Rizky Januar Akbar, S.Kom., M.Eng. selaku pembimbing I yang telah membimbing dan memberikan saran, tugas dan bimbingan yang sangat luar biasa dalam menyelesaikan Tugas Akhir ini.
5. Bapak Ir FX. Arunanto, M.Sc. beserta seluruh dosen Teknik Informatika ITS yang telah membagi dan memberikan ilmu yang bermanfaat untuk penulis selama melakukan studi.
6. Teman penulis yang tergabung dalam Albrotherhood serta teman teman angkatan 14 dan 15 di Laboratorium Alpro

yang senantiasa memberikan pelajaran dan juga memberikan motivasi, kenangan dan kesenangan selama menjalani studi di Informatika .

7. Teman-teman seperjuangan dari anak bimbingan Pak Rizky yang selalu membuat penulis merasa paling ketinggalan sehinggga memacu semangat dalam mengerjakan Tugas Akhir.
8. Teman-teman Jaeger di game DOTA 2 yang senantiasa mengajak, menunggu dan begadang bersama baik ketika find match maupun in-game, menang maupun kalah.
9. Sahabat serta teman maupun orang yang bertemu dengan penulis yang tidak dapat disebutkan satu per satu namanya yang memberikan kesadaran, pelajaran, ilmu maupun tempat bertukar pikiran selama penulis melaksanakan studi.

Semoga dengan adanya Tugas Akhir ini dapat membawa manfaat baik bagi penulis maupun pengguna Tugas Akhir ini kedepannya. Dengan kerendahan hati penulis mengharapkan kritik dan saran yang membangun perbaikan ke depan.

Surabaya, Juli 2017

Andre Zachary Reinaldi

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

(Halaman ini sengaja dikosongkan)

DAFTAR KODE SUMBER

(Halaman ini sengaja dikosongkan)

BAB 1
PENDAHULUAN

1.1 Latar Belakang

Belajar merupakan sebuah kegiatan yang dilakukan oleh individu untuk dapat mengenali dan mengetahui lebih lanjut tentang hal-hal yang berguna. Dalam proses belajar terdapat urutan–urutan. Contohnya dalam ilmu informatika ketika kita belajar tentang program Android, maka kita perlu belajar tentang Java, pemahaman tentang *object oriented* dan sebagainya. Maka dari itu jika terdapat dokumentasi tentang proses belajar sesuai dengan urutan maka akan mempercepat proses pembelajaran.

Selama ini dokumentasi atau pencatatan proses pembelajaran tersebut dalam bentuk statis antara lain buku, tutorial *source code*, makalah dan lain-lain. Tapi terkadang urutan proses hilang terutama dalam hal *source code*, dimana banyak tutorial atau buku menuliskan *code* yang telah selesai secara langsung bukan dalam urutan-urutan. Dalam dokumentasi secara dinamis misalnya video, memang meng-*capture* urutan proses ketika membuat *code*, tetapi orang yang belajar dari video tersebut harus mem-*pause* atau mengulang kembali video jika ada urutan *code* yang terlalu cepat atau tidak dimengerti. Hal ini tentu sangat menghabiskan waktu.

Belajar menjadi lebih efektif ketika terdapat *source code* yang jelas dan terdokumentasi. Sangat banyak *source code* yang dibuat oleh orang-orang di seluruh dunia. *Source code* ini terdokumentasi dalam *Version Control System* (*VCS*). *Version Control System* memudahkan *coder* karena *VCS* menyimpan tahapan-tahapan *code* dalam bentuk *snapshot* sehingga *coder* lain bisa bekerja sama dengan lebih baik. Namun *snapshot* hanya diambil ketika *coder* melakukan *commit*, jadi ketika *coder* melakukan *commit* hanya saat *code* sudah jadi hanya terdapat *snapshot* yang sedikit. Sehingga proses pembelajaran dari urutan-urutan *code* pun juga hilang. Padahal jika urutan *code* tersebut bisa didapatkan maka banyak sekali manfaat yang diperoleh.

Karena hal di atas, dalam proposal ini diajukan sebuah metode untuk merekam urutan pengerjaan *code*. Dalam penelitian sebelumnya[1], telah dibuat *plug-in* yang hampir sama, tetapi *plug-in* ini hanya dapat merekam *insert update delete* berdasarkan *keystroke*. Sedangkan dalam tugas akhir ini, *plug-in* akan merekam jenis aktivitas apa yang dituliskan.

Dengan metode ini diharapkan proses pengerjaan *code* dapat disimpan berupa *log* aktivitas yang dikerjakan selama *editing*, contohnya ketika membuat sebuah proyek Android maka *log* yang terbentuk adalah *add class, add method*, dan lain sebagainya. *Log* aktivitas tersebut akan disimpan dalam sebuah *file* tersendiri. Kemudian *file* tersebut dapat digunakan untuk tutorial kepada peserta didik agar proses pembelajaran efektif.

1.2 Rumusan Masalah

Tugas akhir ini mengangkat beberapa rumusan masalah sebagai berikut:

1. Bagaimana merekam *keystroke* pada *Intergrated Development Environment*?
2. Bagaimana mendefinisikan aktivitas-aktivitas dalam proses pemrograman?
3. Bagaimana memproses *keystroke* menjadi sebuah *log* aktivitas?

1.3 Batasan Permasalahan

Permasalahan yang dibahas pada tugas akhir ini memiliki batasan sebagai berikut:

1. Bahasa pemrograman yang akan digunakan adalah bahasa pemrograman utama Java.
2. Pembangunan aplikasi dilakukan menggunakan IDE Eclipse dengan menggunakan Eclipse *development plugin*.
3. Orientasi pemrograman yang digunakan bersifat *Object oriented*.
4. *Plug-in* yang dibangun hanya dapat mendeteksi perubahan melalui *keystroke keyboard*.

1.4 Tujuan

Tugas akhir ini bertujuan untuk merekam aktivitas pemrograman yang dilakukan oleh programmer dengan cara mendapatkan *keystroke* dari *user* saat *edit* kode, kemudian aktivitas pemrograman tersebut akan diklasifikasikan. Aktifitas pemrograman yang telah terklasifikasikan tersebut akan dicatat secara urut.

1.5 Manfaat

Tugas akhir ini dapat menjadi dokumentasi pembelajaran atau tutorial untuk pemula membuat kode dari nol sampai tuntas. Kedepannya *log* aktivitas *code* yang banyak dapat digunakan untuk *data mining* sehingga dapat mengoptimasi aktivitas pemrograman, Contohnya seperti proses *autocomplete* yang dimodifikasi lebih lanjut dan lain sebagainya.

1.6 Metodologi

Pembuatan tugas akhir ini dilakukan dengan menggunakan metodologi sebagai berikut:

1.6.1 Penyusunan Proposal Tugas Akhir

Tahap pertama dalam proses pengerjaan tugas akhir ini adalah menyusun proposal tugas akhir. Pada proposal tugas akhir ini diajukan sebuah program tambahan atau *plug-in* di Eclipse IDE untuk membuat *log* aktivitas urutan proses *code*.

1.6.2 Studi literatur

Tahapan ini diisi dengan pencarian studi literatur yang relevan untuk dijadikan referensi dalam pengerjaan tugas akhir. Studi literatur ini didapatkan dari buku, internet, dan materi-materi kuliah yang berhubungan dengan metode yang akan digunakan.

1.6.3 Analisis Desain Perangkat Lunak

Perangkat Lunak yang dibangun merupakan *plug-in* IDE Eclipse berbasis *desktop* Windows. *Plug-in* merupakan Maven

plug-in project yang menggunakan *library* dari Stateforge untuk membuat representasi *finite state machine*.

1.6.4 Pengembangan Perangkat Lunak

Pembangunan *plug-in* ini dilakukan dengan bahasa pemrograman Java, IDE Eclipse, Maven, Stateforge untuk *library* yang digunakan.

1.6.5 Pengujian dan Evaluasi

Pengujian dilakukan oleh beberapa orang *programmer*. *Programmer* akan menulis *code* yang beragam seperti contohnya membuat *class*, membuat *method*, menginisialisasi *variable* dan sebagainya. Sebuah *file* akan dibuka untuk memverifikasi apakah semua aktivitas pemrograman yang dilakukan sudah terklasifikasikan dengan benar. Lingkungan pengujian ini hanya dilakukan di IDE Eclipse, dan membuatnya secara bertahap.

1.6.6 Penyusunan Buku Tugas Akhir

Pada tahap ini dilakukan penyusunan laporan yang menjelaskan dasar teori dan metode yang digunakan dalam tugas akhir ini. Pada tahap ini juga disertakan hasil dari implementasi metode yang telah dibuat.

1.6.7 Sistematika Penulisan Laporan

Sistematika penulisan laporan tugas akhir adalah sebagai berikut:

1. Bab I. Pendahuluan
   Bab ini berisi penjelasan mengenai latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat, metodologi, dan sistematika penulisan dari pembuatan tugas akhir.
2. Bab II. Tinjauan Pustaka
   Bab ini berisi kajian teori dari metode dan algoritma yang digunakan dalam tugas akhir ini. Secara garis besar bab ini berisi tentang *plug-in development*, *finite state machine*, *document event*, *generator state machine*.

3. Bab III. Perancangan Perangkat Lunak
   Bab ini berisi pembahasan mengenai perancangan dari metode penangkapan *keystroke event* dari *programmer* ketika mengetikkan *code*, cara penulisan kode oleh *programmer* dan desain dari *state machin*e yang diimplemetasikan.
4. Bab IV. Implementasi
   Bab ini menjelaskan implementasi berbasis diagram dan juga *pseudocode* untuk metode pengambilan *keystroke* pada IDE dan juga mendefinisian aktivitas pemrograman berdasarkan *state machine*.
5. Bab V. Hasil Uji Coba dan Evaluasi
   Bab ini berisi hasil uji coba dari *plug-in*. Uji coba dilakukan dengan cara merekam aktivitas pemrograman yang dibuat oleh *programmer* dengan menggunakan IDE Eclipse yang sudah ter-*install plug-in* yang telah dibuat. Hasil evaluasi mencakup apa saja yang ditangkap *plug-in* mengenai aktivitas pemrograman yang telah tercatat di dalam *file log*.
6. Bab VI. Kesimpulan dan Saran
   Bab ini merupakan bab yang menyampaikan kesimpulan dari hasil uji coba yang dilakukan, masalah-masalah yang dialami pada proses pengerjaan tugas akhir, dan saran untuk pengembangan *plug-in* kedepannya.
7. Lampiran
8. Daftar Pustaka
   Bab ini berisi daftar pustaka yang dijadikan literatur dalam tugas akhir.

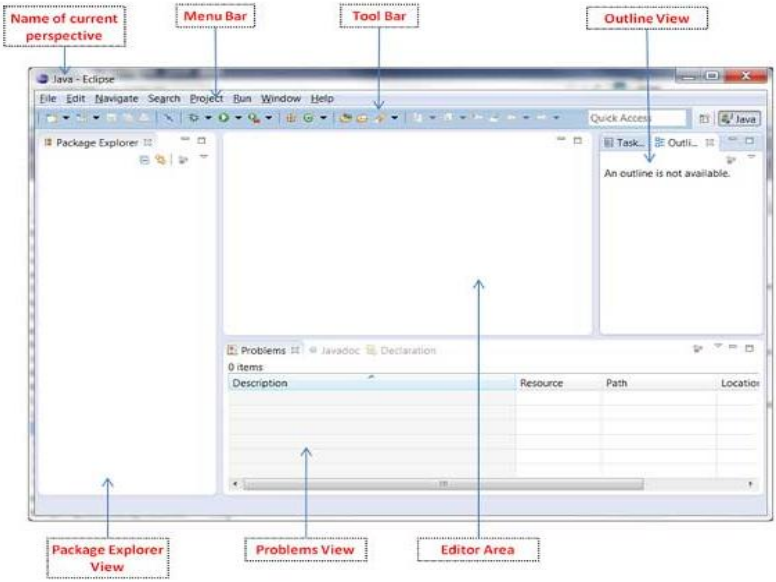
(Halaman ini sengaja dikosongkan)

BAB 2
TINJAUAN PUSTAKA

Dalam tugas akhir ini, terdapat beberapa tinjauan pustaka yang digunakan, yaitu sebagai berikut.

2.1 Eclipse

Eclipse merupakan sebuah *Intergrated Development Environtment*. Eclipse dirancang untuk membangun sistem web yang terintegrasi dan alat untuk pengembangan aplikasi. Berdasarkan rancangan, Eclipse tidak menyediakan fungsionalitas *end user* yang baik secara *stand-alone*. Namun nilai dari *platform* Eclipse yang diunggulkan yaitu pengembangan yang cepat dari fitur-fitur yang terintegrasi berdasarkan model *plug-in*[2]. Eclipse memiliki sebuah *workSpace* utama dan sebuah sistem *plug-in* yang dapat ditambahkan untuk meng-*custom environtment* sistem.

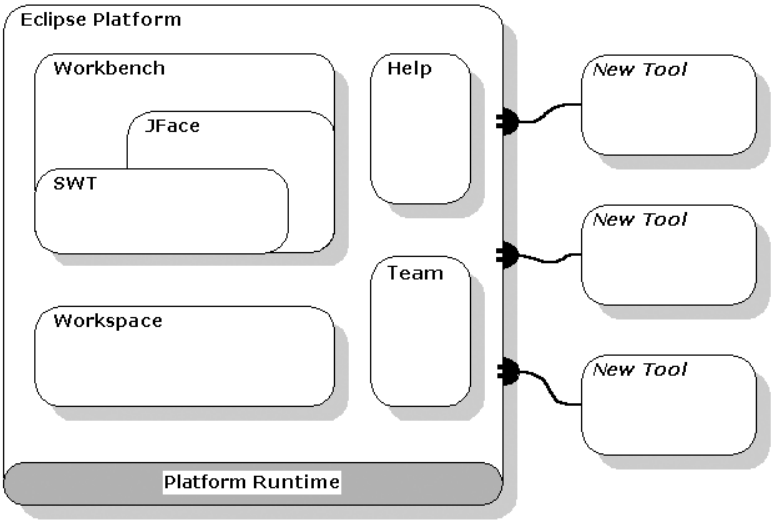


Gambar 2.1 Eclipse *window*

Sebuah *window* Eclipse memiliki empat part besar yaitu[3]: *views*, *editor*, *menu bar* dan *toolbar*. Sebuah Eclipse *window* bisa memiliki banyak *editor* dan *view*, tetapi hanya satu saja yang aktif dalam satu waktu. Secara umum *editor* digunakan untuk melakukan *editing* data project dan *view* digunakan untuk melihat *metadata* dari project.

2.2 Eclipse Plugin Architecture

A plug-in in Eclipse is a component that provides a certain type of service within the context of the Eclipse workbench [4]. *Plug-in* tidak terdapat langsung dalam distribusi dari *package* Eclipse tetapi harus ditambahkan secara *partial* tergantung *plug-in* apa yang dibutuhkan. *Plug-in* akan mengambil data dari Eclipse *workbench* dan mengolah data itu sesuai *service* yang diberikan oleh *plug-in* tersebut. Sebuah Eclipse *workbench* dapat memiliki banyak *instace plug-in* seperti pada Gambar 2.2.

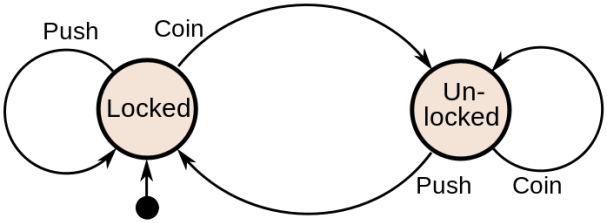


Gambar 2.2 Arsitektur *plug-in* sistem

Dalam Eclipse terdapat sebuah *instance* Eclipse *runtime* yang menyediakan infrastruktur untuk melakukan *support* aktivasi dan operasi dari *plug-in set* yang bekerja bersama untuk menyediakan lingkungan *development*.

2.3 Finite State Machine

Sebuah *finite state machine* secara definisi adalah sebuah model matematika dari komputasi. Sebuah FSM dapat berpindah dari satu *state* ke *state* lainnya karena *input external* yang diberikan ke dalam *state machine* tersebut. Perpindahan tersebut dinamakan transisi. Sebuah FSM biasanya didefinisikan oleh *state-state* yang ada serta kondisi-kondisi untuk transisi setiap *state* tersebut.

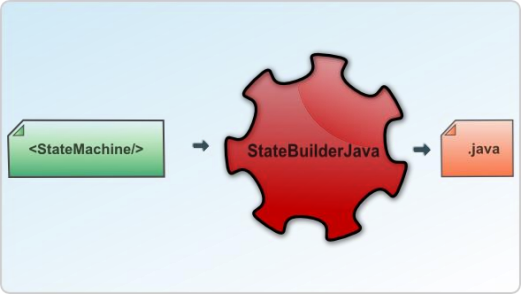


Gambar 2.3 *State machine* sederhana

Pada gambar contoh *state* yang ada adalah *Locked* dan *Unlocked*. Transisi pada *Locked state* saat diberikan *input push* adalah kembali ke *Locked state*, saat diberikan *input coin* maka *state machine* pindah ke *unlocked state*. Pada *Unlocked state* saat diberi *input coin* maka akan kembali ke *Unlocked state* saat *push* maka ke *Locked state*.

2.4 StateForge

StateForge[5] merupakan *state machine code generator*. StateBuilderJava adalah salah satu *library* yang digunakan untuk men-*generate code* dengan Bahasa Java. StateBulderJava merubah sebuah deskripsi *statemachine* ke dalam versi *code* Java.



Gambar 2.4 Cara kerja *library* StateForge

Deskripsi *statemachine* dituliskan dalam sebuah *file xml yang* memiliki *extensi* fsmjava. *File* tersebut memiliki *schema* seperti pada Kode Sumber 2.1.

Element setting memiliki *attribute asynchronous* yang menspesifikasikan apakah *state machine* yang dibuat *synchronous* ataupun tidak. *NameSpace* untuk menspesifikasikan nama *class* dari *state machine*. *Element object* untuk menspesifikasikan *class* yang digunakan sebagai *external resource* dari *state machine*. *Element eventSource* untuk menspesifikasikan *event* apa saja yang ada di *state machine*. *Element event* memiliki *atribut id* yang mendefinisikan nama *event* tersebut. *Element state* memiliki atribut *name* untuk menspesifikasikan nama *state*. Di dalam *element state* terdapat *element onEntry*, *transition*, *onExit*, *condition*. *Element onEntry* dan *onExit* memiliki *atribut action* untuk menspesifikasikan apa yang dilakukan *state* saat *state machine* memasuki atau keluar dari *state* tersebut. *Element transition* memiliki atribut *event* yang berarti *event* yang memicu

terjadinya *transition* dan *atribut nextState* yang menspesifikasikan *state* selanjutnya.

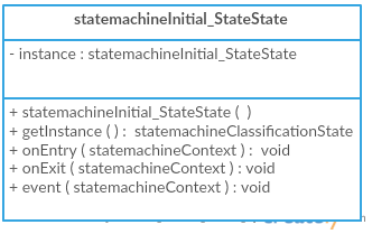
```
1.  <sm:StateMachine . . . >
2.      <!-- General settings -->
3.      <settings asynchronous="false" nameSpace="
    Classification">
4.          <object instance="action" class="Action"/>
5.      </settings>
6.      <!-- Events -->
7.      <events>
8.          <eventSource name="Event">
9.              <event id="event"/>
10.         </eventSource>
11.     </events>
12.     <!-- States -->
13.     <state name="Classification">
14.         <state name="Initial_State">
15.             <onEntry action=". . . "/>
16.             <transition event="event" nextState="."/
    >
17.         </state>
18.     </state>
19. </sm:StateMachine>
```

Kode Sumber 2.1 XML *Schema* dari *state machine*

Berdasarkan deskripsi *state* dari XML tersebut di-*generate file* Java. *File* Java yang dihasilkan adalah statemachineContext, statemachineClassificationState dan *class* dari *state* yang didefinisikan dalam *element* state di xml sperti yang ditunjukan di Gambar 2.5. Dalam *class state* terdapat *method event* yang didefinisikan dalam *element transition*.

Untuk menggunakan *state machine* yang telah di-*generate*, program harus membuat sebuah *class* yang berisi *object* dari *state machine* dan memiliki *method* bernama *event* yang dispesifikasikan sebelumnya. Dalam merepresentasikan *input*,

program memanggil *method event* dari *object state machine* sehingga terjadi transisi dari *state*.



Gambar 2.5 *Class* hasil *generate state machine*

2.5 Change Impact Analysis

"Change Impact analysis, a collection of techniques for determining the effect of a set of source code changes"[6]. Change impact analisis adalah bagaimana sebuah perubahan dari code dapat memberikan efek pada *code* yang lain. Perubahan yang dituliskan diubah menjadi berberapa set perubahan yang *atomic*. Perubahan yang *atomic* itu didefinisikan dalam Tabel 2.1.

Tabel 2.1 Kategori dari perubahan yang *atomic*

(AC)	Add an empty clas
(DC)	Delete an empty class
(AM)	Add an empty method
(DM)	Delete an empty method
(CM)	Change body of method
(LC)	Change virtual method lookup
(AF)	Add a field
(DF)	Delete a field

Dalam tugas akhir ini *change* dari *code* menjadi inti dari *plug-in* bukan pada *impact analysis*. Perubahan yang dicatat merupakan hasil dari *keystroke user* dalam *program*. Kemudian *keystroke* itu akan dianalisis apakah *keystroke* tersebut memiliki makna tertentu. Makna yang dimaksud yaitu antara lain *add class*,

add method, *add variable*, *call method* dan aktivitas lain yang dijelaskan dalam subbab selanjutnya dalam buku ini. Perubahan seperti *delete a field* tidak ada dalam tugas akhir ini dikarenakan metode dalam pengambilan *keystroke* yaitu documentListener tidak mendapatkan informasi mengenai *text* apa yang dihapus.

2.6 Bahasa Pemrograman Java

Java adalah bahasa pemrograman umum yang di-*compile* sebagai kesatuan, berbasis *class* dan berorientasi *object* dan didesain spesifik untuk memiliki ketergantungan yang sesedikit mungkin. Java diinisialisasi oleh James Gosling, Mike Sheridan dan Patrick Naughton pada juni 1991. Sampai sekarang sudah keluar 9 versi dari java language, terakhir JAVA SE 8 (March 8, 2014)[7].

Java seperti pemrograman lain berdasar pada *syntax* dan *grammar* yang dikembangkan oleh Bells Lab pada tahun 1960. Bahasa pemrograman umumnya sama seperti bahasa sehari-hari yang kita gunakan terdapat aturan-aturan, contohnya dalam Bahasa Indonesia sebuah kalimat minimal terdapat subjek dan predikat. Tanpa itu maka tidak bisa disebut sebuah kalimat.

Dalam Bahasa Java terdapat *lexical structure*, yaitu aturan yang menspesifikasikan bagaimana program dituliskan. Aturan-aturan tersebut antara lain sebagai berikut[8].

- Java itu *case sensitive*.
- *WhiteSpace* diabaikan.
- Statement berakhir di titik koma(“;”).
- *Reserved keywords* yang memiliki arti tersendiri dalam *syntax* Bahasa Java.
- *Literals* adalah sebuah data konstan, dapat berbentuk *integer*, *string* atau *karakter*.
- *Identifier* adalah nama dari *variable* dan fungsi. Tidak boleh kata kata yang masuk dalam *keywords*.

Contoh dalam mendeklarasikan sebuah aktivitas perograman dalam Java memiliki *syntax* yang digunakan dalam tugas akhir adalah sebagai berikut[9].

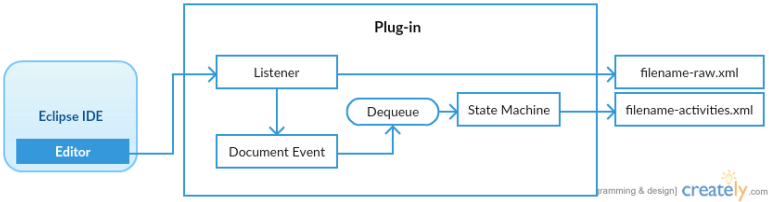
- *Class Declaration*
  {ClassModifier} class Identifier [type parameters] [Superclass] [SuperInterfaces] ClassBody.
- *Field*/*Variable Declaration*
  {Modifier} tipedata identifier;
- *Field*/*variable initialization*
  {Modifier} tipedata identifier = variableinitializer;
- *Object Instantiation*
  New identifier(Parameter);

- *Method Declaration*
  {MethodModifier} ResultType Identifier(Parameter){};
- *Calling* a method
  Identifier(Parameter);
- *Superclass*/ *Extends* a *class*
  Extends ClassType.
- *SuperInterfaces*/ *implements an Interface*
  Implements InterfaceTypelist.

BAB 3
PERANCANGAN PERANGKAT LUNAK

3.1 Arsitektur Sistem

Secara garis besar arsitektur sistem yang dibuat seperti Gambar 3.1.

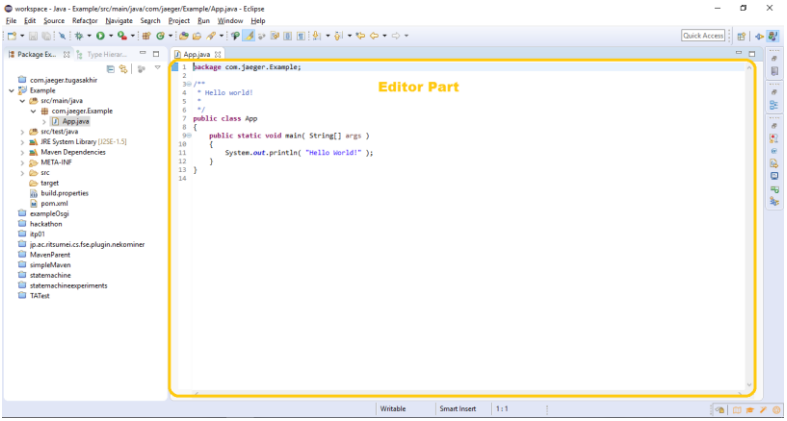


Gambar 3.1 Arsitektur *plug-in*

Plug-in akan menambahkan *listener* pada *editor* sehingga ketika *user* menuliskan *code program* maka *listener* akan mendapatkan perubahan pada *editor* dalam bentuk *DocumentEvent*. *DocumentEvent* tersebut akan dimasukkan ke dalam *dequeue*. Ketika terdapat *end character* seperti titik koma(“;”) ataupun kurung kurawal tutup (“}“), isi dari *dequeue* akan dimasukkan ke dalam *state machine* untuk diklasifikasikan dan dikeluarkan dalam bentuk *file-file* xml

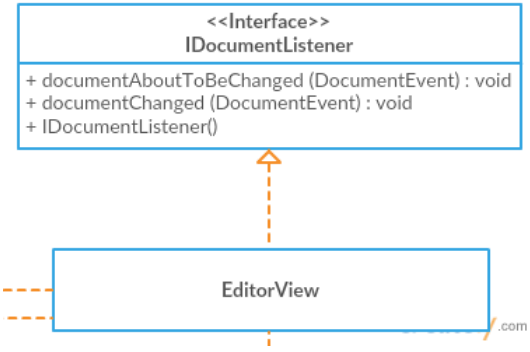
3.2 *Keystroke Event Listener*

Plug-in yang dibuat berjalan di atas IDE Eclipse. Struktur pada IDE Eclipse seperti berikut



Gambar 3.2 IDE Eclipse

Di dalam *workbench* terdapat *perspective*, di dalam *perspective* tersebut terdapat *editor*. *Plug-in* akan menambahkan *IDocumentListener* pada *editor* yang aktif sehingga perubahan yang terjadi akan tercatat.



Gambar 3.3 Class diagram untuk IDocumentListener

Listener berfungsi untuk mengambil semua perubahan yang terjadi pada dokumen, tetapi *listener* bekerja secara *context*

free. *Listener* tidak tahu isi dokumen sepenuhnya seperti apa, tidak tahu dimana letak *text* sebelum maupun sesudahnya ataupun *history* pengetikan sebelumnya.

Karena *plug-in* mencatat perubahan dari dokumen dengan *listener* pada *editor*, *plug-in* tidak mendapatkan perubahan yang dibuat saat ada bagian kode yang dibuat dengan *wizard*. *Wizard* yang termasuk di sini adalah ketika *wizard* membuat sesuatu yang belum terbuka sebelumnya dalam *editor* aktif. Contohnya ketika membuat *file* → *new* → *class* dengan *wizard*, *file* yang terbentuk didalamnya sudah berisi sebuah *Class*. *Class* ini tidak akan tercatat oleh *plug-in*.

3.3 Analisis Perilaku Pemrograman

Membahas mengenai *event*/perilaku (*behaviour*) *programmer* ketika mengetikkan kode pada IDE/ *editor*. IDE mencatat perubahan pada dokumen dengan *documentChangeListener* yang mengembalikan *value* dengan tipe *DocumentEvent* yang berisi *offset*, *length*, *timestamp* dan *text*. *Offset* adalah tempat dimana perubahan terjadi dalam dokumen. *Length* menunjukkan jumlah dokumen yang tergantikan. *Timestamp* menunjukkan waktu perubahan terjadi. Text menunjukkan penambahan *text* pada dokumen.

1. *Character by character* (*Top to Down – Left to Right*)

Di sini *user* memasukkan data secara sequensial tanpa ada perpindahan kursor. IDE mencatat perubahan yang ada di dokumen seperti berikut,

```
>> offset: 53, length: 0, timestamp: 326 text:>r<
>> offset: 54, length: 0, timestamp: 327 text:>i<
>> offset: 55, length: 0, timestamp: 328 text:>v<
>> offset: 56, length: 0, timestamp: 329 text:>a<
>> offset: 57, length: 0, timestamp: 330 text:>t<
>> offset: 58, length: 0, timestamp: 331 text:>e<
```

Gambar 3.4 Hasil *output listener character by character*

DocumentChangeListener mencatat *offset* yang berurutan dari perubahan document hal ini menunjukkan karakter yang dituliskan secara sequensial ke dalam dokumen. *DocumentEvent* selalu berisi satu karakter dan *length* nol, tetapi *behavior* dari *DocumentChangeListener* ketika mencatat *offset* adalah tidak mengetahui letak dimana penulisan perubahan, *offset* mencatat secara *sequence* meskipun *user* menuliskan kode satu *line* di atas kode sebelumnya.

Sebagai contoh saat *user* menuliskan *private int* dengan *int* dituliskan 1 *line* di atas *private* seperti berikut

int ← i memiliki offset 8 n offset 9 dan t offset 10.

private ← karakter e memiliki offset 7.

Jadi *DocumentChangeListener* tidak mencatat letak dari perubahan teks di dokumen.

2. *Autocomplete* (*Ctrl + Space*)

Untuk *behavior autocomplete* ini *user* menggunakan *tool* dari IDE untuk melengkapi kode yang baru sebagian dituliskan sebagai contoh di sini *user* menuliskan pri kemudian menekan *Ctrl + Space* untuk membuka *plug-in autocompletion* dan memilih *private*. *DocumentChangeListener* mendapatkan *output* seperti berikut

```
>> offset: 52, length: 0, timestamp: 260 text:>p<

>> offset: 53, length: 0, timestamp: 261 text:>r<

>> offset: 54, length: 0, timestamp: 262 text:>i<

>> offset: 52, length: 3, timestamp: 263
text:>private<
```

Gambar 3.5 Hasil *output listener autocomplete*

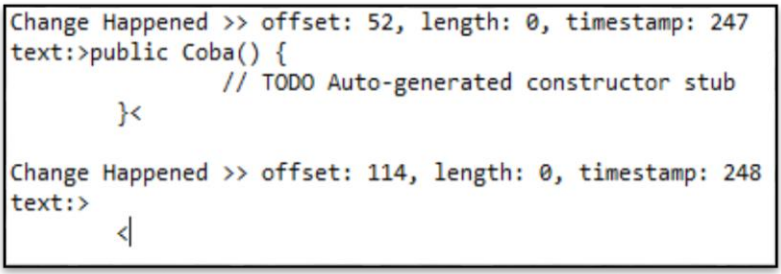
Untuk *autocomplete* setelah kita menuliskan beberapa kata kemudian menekan *Ctrl + Space* akan keluar *suggestion*, setelah kita pilih maka *documentChangeListener* akan mencatat perubahan di *offset* huruf awal dengan *length* dari keseluruhan perubahan serta isi *text* yang penuh dari awal sampai akhir.

```
 >> offset: 169, length: 0, timestamp: 524 text:>S<

 >> offset: 170, length: 0, timestamp: 525 text:>y<

 >> offset: 171, length: 0, timestamp: 526 text:>s<

 >> offset: 172, length: 0, timestamp: 527 text:>o<

 >> offset: 173, length: 0, timestamp: 528 text:>u<

 >> offset: 174, length: 0, timestamp: 529
text:>t<

 >> offset: 169, length: 6, timestamp: 530
text:>System.out.println();<
```

Gambar 3.6 Hasil *output listener autocomplete 2*

3. Template

Untuk template *user* tidak mengetikkan kode terlebih dahulu dan langsung menekan *Ctrl + Space* maka akan muncul *window*. *User* hanya perlu memilih mana yang akan diimplementasikan.



Gambar 3.7 Hasil *output listener template*

Dengan menekan *Ctrl + Space*, muncul *suggestion* ketika dipilih maka *documentChangeListener* langsung mencatat seluruh perubahan tapi *length* tetap 0. Hal ini berbeda dengan *autocomplete* yang telah disebutkan sebelumnya. Karena dalam *template* ini *programmer* tidak mengetikkan apa-apa sehingga tidak ada pergantian kode yang telah ditulis sebelumnya.

4. *Deletion* (*BackSpace* / *delete*)

Backspace

Untuk menghapus menggunakan *backspace*, *user* mengetikkan sesuatu ke dalam dokumen kemudian menghapus dari posisi kursor terakhir ke posisi kursor 1 karakter di sebelah kiri.

Pada Gambar 3.8 *DocumentListener* mencatat perubahan *text*. *offset* yang tertangkap semakin berkurang, *length* menunjukkan berapa banyak yang *character* yang berubah dan *text* yang menggantikan berupa *null* (kosong).

Delete :

Untuk menghapus menggunakan *delete*, teks yang dihapus berada 1 karakter di sebelah kanan kursor.

Delete menghapus *text* yang berapa di kanan kursor sehingga perubahan yang terjadi di *offset* yang sama. *Length* dan *text* sama seperti yang terjadi apabila menggunakan *backSpace*.

```
>> offset: 72, length: 0, timestamp: 341 text:>p<
>> offset: 73, length: 0, timestamp: 342 text:>u<
>> offset: 74, length: 0, timestamp: 343 text:>b<
>> offset: 75, length: 0, timestamp: 344 text:>l<
>> offset: 76, length: 0, timestamp: 345 text:>i<
>> offset: 77, length: 0, timestamp: 346 text:>c<
>> offset: 77, length: 1, timestamp: 347 text:><
>> offset: 76, length: 1, timestamp: 348 text:><
>> offset: 75, length: 1, timestamp: 349 text:><
```

Gambar 3.8 Hasil *output listener backSpace*

5. Undo redo

Undo

Undo berfungsi untuk mengembalikan dokumen ke *state* sebelumnya, bukan satu karakter melainkan *state* dari dokumen tersebut.

```
>> offset: 102, length: 1, timestamp: 470 text:><
>> offset: 102, length: 1, timestamp: 471 text:><
>> offset: 102, length: 1, timestamp: 472 text:><
>> offset: 102, length: 1, timestamp: 473 text:><
```

Gambar 3.9 Hasil *output listener delete*

Saat *undo* (*Ctrl* + Z), perubahan terjadi di *offset* awal di mana *state* sebelumnya mulai dan *length* yang terganti sebanyak perubahan dari *state* terbaru ke *state* sebelumnya.

```
>> offset: 72, length: 0, timestamp: 417 text:>t<
>> offset: 73, length: 0, timestamp: 418 text:>r<
>> offset: 74, length: 0, timestamp: 419 text:>y<
>> offset: 75, length: 0, timestamp: 420 text:> <
>> offset: 76, length: 0, timestamp: 421 text:>u<
>> offset: 77, length: 0, timestamp: 422 text:>n<
>> offset: 78, length: 0, timestamp: 423 text:>d<
>> offset: 79, length: 0, timestamp: 424 text:>o<
>> offset: 80, length: 0, timestamp: 425 text:> <
>> offset: 81, length: 0, timestamp: 426 text:>r<
>> offset: 82, length: 0, timestamp: 427 text:>e<
>> offset: 83, length: 0, timestamp: 428 text:>d<
>> offset: 84, length: 0, timestamp: 429 text:>o<
>> offset: 85, length: 0, timestamp: 430 text:>;<

>> offset: 72, length: 14, timestamp: 416 text:><
```

Gambar 3.10 Hasil *output listener undo*

Redo (*Ctrl* + Y)

Redo berkebalikan dari *undo* yaitu mengembalikan *state* setelahnya jika ada.

```
 >> offset: 72, length: 0, timestamp: 430
text:>try undo redo;<
```

Gambar 3.11 Hasil *output listener redo*

Saat *redo text* ditangkap seluruhnya dan karena tidak ada pergantian pada *document* maka *length* 0.

6. Copy paste

Untuk *copy* dan *paste*, *documentChangeListener* hanya memerhatikan *paste action*, karena hanya pada *paste* dokumen berubah

```
 >> offset: 68, length: 1, timestamp: 431
text:> private long offset;<
```

Gambar 3.12 Hasil *output listener copy-paste*

Saat *copy-paste*, *text* ditangkap semua dan sebagai sebuah kesatuan *length* tapi memiliki nilai 1, *offset* mencatat tempat dimana *paste* dilakukan.

7. Autogenerate

Autogenerate terjadi ketika *user* melalui *wizard*, membuat sesuatu. Contohnya ketika ada *variable* yang ingin dibuat *getter* mapun *setter* secara otomatis. Selama *editor* aktif masih membuka *file* yang berubah, maka *listener* akan mendapatkan hasil yang mirip dengan *autocomplete*.

8. Middle insertion

Middle insertion adalah ketika *user* mengetikkan sesuatu saat kursor berada di tengah-tengah teks yang sudah ada sebelumnya.

Sama seperti *character by character* biasa tetapi yang berbeda terdapat pada *offset* terjadinya.

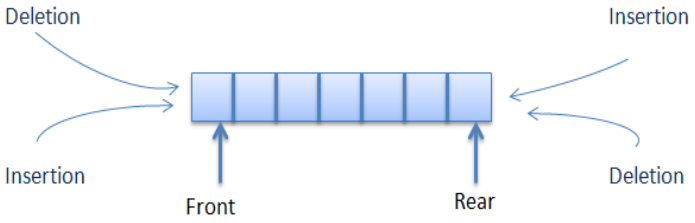
```
>> offset: 60, length: 0, timestamp: 463 text:>i<
>> offset: 61, length: 0, timestamp: 464 text:>n<
>> offset: 62, length: 0, timestamp: 465 text:>t<
>> offset: 60, length: 0, timestamp: 463 text:>s<
>> offset: 61, length: 0, timestamp: 464 text:>t<
>> offset: 62, length: 0, timestamp: 465 text:>a<
>> offset: 63, length: 0, timestamp: 466 text:>t<
>> offset: 64, length: 0, timestamp: 467 text:>i<
>> offset: 65, length: 0, timestamp: 468 text:>c<
>> offset: 66, length: 0, timestamp: 469 text:> <
```

Gambar 3.13 Hasil *output listener middle insertion*

3.4 Desain *Double Ended Queue*

"Double Ended Queue (*dequeque, deque* atau *deck*) *is a linear that supports element insertion and removal at both ends"*[10]. Terdapat 2 ujung di *dequeue* yaitu *front* dan *rear*, representasi *dequeue* ditunjukkan dalam Gambar 3.14.

Dequeue dibutuhkan untuk membuat *stream input* yang dimasukkan oleh *programmer* dapat dirubah ke bentuk *final* sehingga ketika implementasi *state machine* menjadi lebih mudah karena *state machine* tidak perlu menangani ketika terjadi *event deletion*, *copy-paste*, *autocomplete* maupun semua event yang sudah disebutkan pada subbab 3.3.



Gambar 3.14 Representasi sebuah *double ended queue* atau *dequeue*

Kejadian-kejadian itu sudah ditangani dalam *dequeue*. Proses masuknya karakter ke dalam *dequeue* ditentukan berdasarkan *behavior* dari *documentChangeListener* ketika mendapatkan *input* dari *programmer*.

Namun tentunya ada saat di mana *dequeue* harus di-*pop* agar karakter yang sudah masuk ke dalam *dequeue* dapat diproses. Penulis tugas akhir ini menggunakan titik koma dan juga kurung kurawal tutup sebagai tanda bahwa *string* yang ada dalam *dequeue* sudah siap untuk diproses.

3.5 Aktivitas Pemrograman

Aktivitas pemrograman yang didefinisikan dalam tugas akhir ini dibagi menjadi dua yaitu *statement* dan *block*. Untuk aktivitas pemrograman berbentuk *block* yaitu *if statement*, *else statement*, *switch statement*, *case statement*, *for statement*, *try statement*, *catch statement*, *return statement*, *do while statement*. Aktivitas pemrograman yang berbentuk *block* tidak akan dimasukkan ke dalam *state machine* karena *statement* tersebut sudah ditulis secara langsung, tidak ditambahi oleh *keyword* yang lain.

Tabel 3.1 Aktivitas pemrograman berbentuk statement

Nama Aktivitas	Code	Contoh	Grammar
Add an empty method	AM	private void xx(){}	Keyword + identifier +()+ { }
Add an empty class	AC	private class MyClass{}	class + identifier + {}
Add a variable/field	AV	int x;	Keyword + identifier + ;
Initialize a variable	IV	x = 15;	Identifier + = + identifier ;
Call a method	CM	xx();	Identifier + ()+ ;
Instansiate an Object	IO	X = new xx();	Identifier + = + new + identifier + ();
Extend a class	EC	extends Application	Extends + identifier
Implement an Interface	II	implements listener	Implements + identifier

Untuk aktivitas pemrograman berbentuk *statement* yang ada di Tabel 3.1 maka akan dimasukkan ke dalam *state machine*

untuk diklasifikasikan. Karena di dalam *statement* terdapat banyak kombinasi antara *keyword, identifier* dan lain sebagainya.

3.6 *State Machine Design*

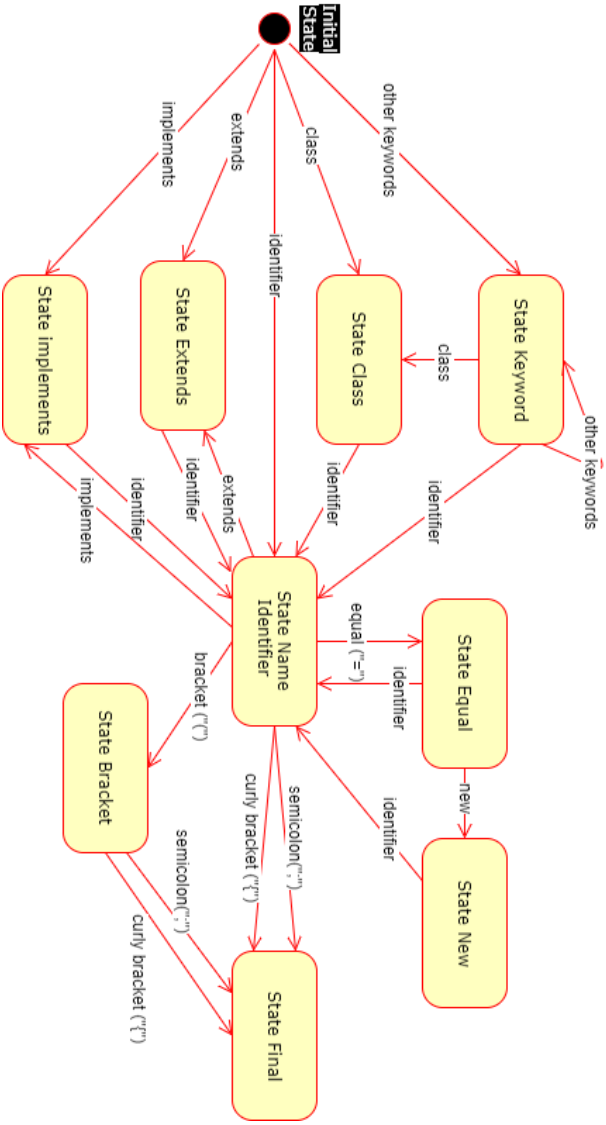
State machine yang digunakan dalam tugas akhir ini menggunakan *library* statemachine1.1.jar dari StateForge[5]. *Library* ini digunakan untuk membuat representasi dari *finite state machine* yang dibuat dalam format *file* .xml.

Untuk mendefinisikan *state machine* yang digunakan untuk *plug-in*, maka perlu dilakukan analisis komponen dari *statement*. Untuk setiap *statement* terdiri atas *token-token*. Dalam pemrograman *token* itu dapat dikategorikan menjadi empat macam :

1. *Identifier*, yaitu *string of letter or digits*.
2. *Integer*, yaitu *non empty string*.
3. *Keyword*, yaitu private, if else, maupun yang sudah didefinisikan oleh Java.
4. *WhiteSpace*, yaitu *a non-empty sequence of blanks, newline* dan *tabs*.

Dalam *state machine* yang digunakan dalam tugas akhir ini berorientasi pada *keyword* karena *input* dalam *state machine* berupa *word*.

Diagram dari *finite state machine* yang digunakan pada tugas akhir ini adalah seperti Gambar 3.15.



Gambar 3.15 Diagram *state machine*

State machine dispesifikasikan dalam **Tabel 3.2**

Tabel 3.2 Tabel state transisi

Current State	Input	Next State
Initial State	Other keyword	Keyword State
	Class	Class state
	extends	Extends State
	implements	Implement State
	identifier	Name Identifier State
Keyword State	class	Class State
	Other keyword	Keyword state
	identifier	Name Identifier State
Class State	identifier	Name Identifier State
Name Identifier State	brackets ("(")	Bracket State
	curly brackets ("{")	Final State
	titik koma (";")	Final State
	extends	Extends State
	Implement	Implements State
	equals ("=")	Equal State
Equal state	identifier	Name Identifier State
	new	New State
Extends State	identifier	Name Identifier State
Implement State	identifier	Name Identifier State

Tabel 3.2 Tabel state transisi

Current State	Input	Next State
New State	identifier	Name Identifier State
Bracket State	identifier	Bracket State
	keyword	Bracket State
	curly brackets ("{")	Final State
	titik koma (";")	Final State
Final State		

3.7 DOM Parser

Document Object Model adalah *interface programming* untuk dokumen HTML dan XML. DOM menyediakan representasi terstruktur dari dokumen sehingga dapat diakses oleh program. DOM merepresentasikan dokumen sebagai sebuah grup yang terdiri atas *nodes* dan *object* yang memiliki *property* dan *method*. *Node* dan *object* merepresentasikan *element* dalam XML *schema*, sedangkan *property* dan *method* merupakan *atribut*. Dalam tugas akhir ini terdapat dua *output file* satu *file* berekstensi name-raw.xml dan yang lain name-activities.xml.

Untuk xml *raw* berisi semua hasil mentah dari *keystroke event* yang dilakukan oleh *programmer*. Bentuk dari -raw.xml adalah seperti berikut:

```
1.  <raw length="0" offset="2" sequence="339">
2.      <text>p</text>
3.  </raw>
```

Gambar 3.16 XML Schema dari filename-raw.xml

Elemen *Root* dari *file* raw.xml adalah *activities*. Di dalam terdapat elemen *raw*, dan *text*. Elemen *raw* memiliki *atribut length*, *offset*, dan *sequence*. *Length* dan *offset* adalah hasil dari *length* dan *offset* saat *DocumentListener* menangkap *event keystroke*, sedangkan *sequence* merupakan urutan *activity*. Elemen *text* berisi *text* dari *DocumentListener*.

activities.xml merupakan hasil klasifikasi *state machine* dengan *input* dari *dequeue* yang sudah dibahas sebelumnya. Bentuk dari -activities.xml dapat dilihat pada Gambar 3.17.

```
1.  <activity sequence="14" type="CM">
2.      <text>public void attackAnother(Pet x);</text>
3.  </activity>
```

Gambar 3.17 XML Schema untuk filename-activities.xml

Root dari *file activities* adalah <activities>. Di dalam terdapat elemen *activity* dan *text*. Elemen *activity* memiliki *atribut sequence* dan *type*. *Sequence* untuk merekam urutan dari *activity*. *Atribute type* untuk merekam hasil klasifikasi *activity* tersebut. Elemen *text* berisi hasil *string output* dari *dequeue*.

(Halaman ini sengaja dikosongkan)

BAB 4
IMPLEMENTASI

4.1 Lingkungan Implementasi

Implementasi *plug-in* untuk klasifikasi aktivitas pemrograman dengan menggunakan *state machine* menggunakan spesifikasi perangkat keras dan perangkat lunak seperti yang ditunjukkan pada Tabel 4.1.

Tabel 4.1 Lingkungan implementasi

Perangkat	Jenis Perangkat	Spesifikasi
Perangkat Keras	Prosesor	Intel(R) Core(TM) i3-3.5 GHz
	Memori	4 GB DDR3
Perangkat Lunak	Sistem Operasi	Windows 10
	Perangkat Pengembang	Eclipse Neon IDE for RCP and RAP Developers

4.2 Implementasi

Dalam mengimplementasikan *plug-in* untuk merekam aktivitas pemrograman menggunakan *finite state machine*. Penulis menggunakan empat pendekatan yaitu menangkap *keystroke input user*, mengimplementasi *dequeue*, mengimplementasi *statemachine*, dan implementasi *klasifikasi*. Kode yang dibuat penulis dapat direpresentasikan dalam *class diagram* yang dilampirkan pada Lampiran A dan Lampiran B.

4.2.1 Implementasi *Keystroke Input User*

Untuk mengimplementasikan *keystroke input user*, *plug-in* harus dimulai segera setelah Eclipse *workbench* siap. Untuk itu Eclipse *plug-in* harus meng-*extend* org.eclipse.ui.startup, *extension point* yang digunakan untuk memanggil *class* EditView saat pertama kali *workbench* dimulai. Ketika *workbench* dimulai *method* earlyStartup() akan dipanggil dalam *thread* yang berbeda.

```
1.  public void earlyStartup() {
2.      IWorkbench wb = PlatformUI.getWorkbench();
3.      ww = PlatformUI.getWorkbench().
4.                          getActiveWorkbenchWindow();
5.      wb.addWindowListener(generateWindowListener());
6.  }
```

Kode Sumber 4.1 *Method* earlyStartup

Method earlyStartup, akan mengambil *workbench window* yang aktif, kemudian akan ditambahkan *window listener* untuk mengetahui *event* yang terjadi di *workbench window* yang aktif.

```
1.  private IWindowListener generateWindowListener() {
2.      // TODO Auto-generated method stub
3.   return new IWindowListener(){
4.   public void windowActivated(IWorkbenchWindow window)
        {
5.          // TODO Auto-generated method stub
6.          IWorkbenchPage activePage = window.
7.                                      getActivePage();
8.          activePage.addPartListener(generateIPartListen
       er2());
9.       }
10. }
```

Kode Sumber 4.2 WindowListener

Dalam *workbench* yang aktif ditambahkan sebuah *part listener* untuk mendeteksi perubahan pada *EditorPart* seperti pada Kode Sumber 4.2. Dalam *part listener* harus diberi sebuah *method*

checkpart seperti pada Kode Sumber 4.4 untuk mengecek apakah *part* dari *workbench* yang aktif sekarang adalah *editor*. *Method checkpart* dipanggil saat *partActivated* ataupun *partOpened*.

Dalam *method checkpart*, jika benar *part* yang aktif sekarang adalah *editor* maka *plug-in* akan membuat sebuah *file* dengan nama *file* yang sama seperti *file* yang dibuka dalam *editor* dengan tambahan -activities maupun -row. Program melakukan hal tersebut dengan memanggil *method* createFile dan melakukan *passing variable* activeprojectname.

Untuk mendapatkan perubahan dokumen ketika *user* melakukan *keystroke* maka ditambahkan *document listener* pada *editor* yang aktif. *Document listener* ini memiliki dua *method* yang harus ada yaitu *method* DocumentChanges dan *method* DocumentAboutToBeChanged. Keduanya memiliki parameter dengan tipe DocumentEvent. DocumentEvent ini berisi perubahan yang terjadi pada dokumen meliputi *offset*, *length*, *timestamp* dan *text*. Pada *IDocumentListener* ini ditambahkan *object dequeue* yang isinya adalah tipe data Document seperti pada Kode Sumber 4.5.

```
1.  IDocumentListener docX = new IDocumentListener(){
2.    Deque<Document> deque = new LinkedList<>();
3.    public void documentAboutToBeChanged(DocumentEvent
    event) {
4.    }
5.    public void documentChanged(DocumentEvent event) {
          dequeOperation(event);
6.    }
```

Kode Sumber 4.3 IDocumentListener

```
1.  IPartListener2 newListener = new IPartListener2(){
2.      private void checkPart(IWorkbenchPartReference par
    tRef){
3.       IWorkbenchPart part = partRef.getPart(false);
4.       if(part instanceof IEditorPart){
5.          title = editor.getTitle();
6.          IEditorInput input = editor.getEditorInput();
7.          IFile file = ((IFileEditorInput) input).
8.                           getFile();
9.          IProject activeProject = file.getProject();
10.         String activeProjectName =
11.                           activeProject.getName();
12.         myFile.createFile(activeProjectName+
13.                           "\\"+title);
14.         if(editor instanceof ITextEditor
15.             && input instanceof FileEditorInput){
16.            IDocument document = (((ITextEditor)editor)
17.                  .getDocumentProvider())
18.                  .getDocument(input);
19.            document.addDocumentListener(docX);
20.          }
21.       }
22.     }
23.     @Override
24.     public void partActivated() {
25.         checkPart(partRef);
26.     }
27.
28.     @Override
29.     public void partBroughtToTop() {
30.         checkPart(partRef);
31.     }
32.
33.     @Override
34.     public void partOpened() {
35.         checkPart(partRef);
36.     }
37. };
```

Kode Sumber 4.4 IPartListener2 yang terdapat method checkPart

```
1.  public class Document {
2.      int offset;
3.      String text;
4.
5.      public Document(int offset, String text) {
6.          super();
7.          this.offset = offset;
8.          this.text = text;
9.      }
10.     public int getOffset() {
11.         return offset;
12.     }
13.     public void setOffset(int offset) {
14.         this.offset = offset;
15.     }
16.     public String getText() {
17.         return text;
18.     }
19.     public void setText(String text) {
20.         this.text = text;
21.     }
22.
23. }
```

Kode Sumber 4.5 Kelas Document

4.2.2 Implementasi *Dequeue*

Setelah perubahan dokumen didapatkan maka perubahan tersebut dimasukkan ke dalam *double ended queue* atau *dequeue*. Hal ini dilakukan untuk mendapatkan hasil *string* akhir yang sudah sesuai dengan *input user* dengan memerhatikan perilaku pemrograman yang telah dijelaskan pada Bab 3.3. Dalam *method* dequeOperation ini dapat dua hal penting yaitu menulis *DocumentEvent* yang terjadi ke dalam namafile-raw.xml dan memasukkan Document Event tersebut ke dalam *dequeue*.

Dequeue akan melakukan *push* maupun *pop* sesuai dengan apa yang dilakukan oleh *user* dan juga karakteristik dari *documentEvent* yang tertangkap oleh *documentChangeListener*.

Sebuah *class Document* dibuat untuk menampung *offset* dan *text* dari *documentEvent* yang tertangkap.

Dequeue akan menyimpan *object Document*. Ketika terdapat *event backSpace* dari *user* maka *dequeue* terakhir akan dikeluarkan, untuk *event keystroke input* maka *string text* yang didapatkan oleh *documentEvent* akan diubah menjadi *string* per karakter dan dibuat *object* sendiri dengan *offset* yang juga menyesuaikan jumlah karakter dari *string* yang dipecah sebelumnya. Ketika terdapat *event* berupa *middle insertion* atau *deletion* maka dilakukan operasi *middle insertion* yaitu *event* yang terjadi sekarang dibandingkan *offset*-nya dengan *document* paling akhir yang ada di *dequeue*, jika *offset* dalam *dequeue* lebih besar maka isi dalam *dequeue* akan dikeluarkan dan ditampung sementara di dalam *array list* sementara, kemudian dilakukan operasi *deletion* atau *insertion*, setelah itu isi dari *array list* sementara dimasukkan ulang ke dalam *dequeue*.

Karena secara garis besar berdasarkan analisis *code behavior* yang didapatkan dari bab 3.3. *code behavior* dapat dibedakan menjadi dua bagian yaitu *deletion* dan *insertion*.

Deletion memiliki spesifikasi dalam *documentEvent* yang didapatkan oleh *documentChangeListerner* yaitu memiliki *length* lebih dari satu dan *text* yang ada panjangnya nol atau tidak ada *text*. Kemudian dicek apakah *event deletion* itu merupakan *middle insertion* atau tidak jika ya maka harus melakukan operasi *middle insertion* yang dijelaskan di atas.

```
1.  if(event.getLength()>=1){
2.      if(deque2.isEmpty()){
3.      }else {
4.        if(event.getLength()==1 &&
5.           event.getText().length()==0){
6.           if(event.getOffset()<=deque2.peekLast().
7.                                      getOffset()){
8.              List<Document> temp = new ArrayList<>();
9.              int move = deque2.peekLast().
10.                     getOffset()- event.getOffset();
11.             for(int i=0;i<move;i++){
12.                temp.add(deque2.removeLast());
13.             }
14.             deque2.removeLast();
15.             for(int j = move-1;j>=0;j--){
16.                Document tempDoc =new Document(
17.                   deque2.peekLast().getOffset()+1,
18.                   temp.get(j).getText());
19.                deque2.add(tempDoc);
20.             }
21.          }else {
22.             deque2.removeLast()
23.          }
24.     }
```

Kode Sumber 4.6 Kode untuk menangani *event deletion*

Jika ternyata *deletion* bukan *delete*/*backSpace* tapi merupakan *autocomplete* atau *event copy paste* maka harus terdapat *handler* tersendiri. Karena *event autocomplete* memiliki *length* yang bervariasi bergantung ada tidaknya karakter yang diganti maka tidak dapat dijadikan parameter. Dicek terlebih dahulu apakah *event* ini menggantikan *document* yang sudah ada di dalam *dequeue*. Jika ya maka dicari mulai dari karakter keberapa, kemudian dokumen terakhir yang ada di dalam *dequeue* akan ditampung ke dalam *variable temp*. Setelah dilakukan pergantian dari karakter menjadi *autocomplete* barulah isi dari *variable temp* dimasukkan lagi ke dalam *dequeue*.

```
1.  else if(event.getOffset()<=deque2.peekLast().getOffse
    t()){
2.     // Autocomplete or undo action
3.     List<Document> temp = new ArrayList<>();
4.     int move = deque2.peekLast().getOffset()
5.              - (event.getOffset()+event.getLength());
6.     if(move>=0){
7.         for(int i=0;i<=move;i++){
8.             temp.add(deque2.removeLast());
9.         }
10.     }
11.     for(int i=0;i<event.getLength();i++){
12.        //replace the string
13.        deque2.removeLast();
14.     }
15.     String x = event.getText();
16.     for(int i = 0;i<x.length();i++){
17.        String in = String.valueOf(x.charAt(i));
18.        if(deque2.isEmpty()){
19.          deque2.add(new Document(
20.                event.getOffset()+i,in));
21.        }else {
22.          deque2.add(new Document(
23.                deque2.peekLast()
24.                    .getOffset()+1,in));
25.         }
26.      }
27.      if(move>=0){
28.         for(int j = move;j>=0;j--){
29.             Document tempDoc =
30.                new Document(deque2.peekLast()
31.                        .getOffset()+1,
32.                    temp.get(j).getText());
33.             deque2.add(tempDoc);
34.          }
35.       }
36.     }
37. }
38.
```

Kode Sumber 4.7 Kode untuk menangani deletion 2

Insertion adalah aktivitas yang tidak melakukan perubahan pada dokumen yang sudah ada sebelumnya. *Insertion* memiliki *length* nol. *Insertion* dibagi menjadi dua bagian, yaitu *insertion* yang merupakan *template* atau *redo* yang memiliki *text length* lebih dari satu tetapi tidak tiga.

Yang dilakukan saat mendapat *event* seperti itu adalah membagi *event text* yang masuk menjadi per karakter dan memasukkannya ke dalam *dequeue* secara berurutan.

```
1.  String x = event.getText();
2.  for(int i = 0;i<x.length();i++){
3.      String in = String.valueOf(x.charAt(i));
4.      deque.add(new Document(event.getOffset()+i,in));
5.  }
```

Kode Sumber 4.8 Mengubah string menjadi karakter

Untuk *insertion* bagian kedua dicek apakah *dequeue* kosong jika ya maka hanya perlu memasukkan.ke dalam *dequeue* karena pasti hanya satu karakter saja. Jika tidak, diperlukan pengecekan apakah merupakan *middle insertion* jika ya perlu dilakukan operasi *middle insertion* yaitu menampung karakter sampai *offset event* tersebut terjadi kemudian menambahkan *event* tersebut ke dalam *dequeue* dan mengembalikan karakter yang sebelumnya ditampung ke dalam *dequeue*. Jika tidak *middle insertion*, maka hanya perlu dimasukkan seperti biasa.

```
1.  if(deque.isEmpty()){
2.      deque.add(new Document(event.getOffset(),
3.                      event.getText()));
4.  }else if(event.getOffset()<=deque.peekLast().getOffse
    t()){
5.      //Autocomplete or undo action
6.      List<Document> temp = new ArrayList<>();
7.      int move = deque.peekLast().getOffset() - event.g
    etOffset();
8.      for(int i=0;i<=move;i++){
9.          temp.add(deque.removeLast());
10.     }
11.     String x = event.getText();
12.     for(int i = 0;i<x.length();i++){
13.         String in = String.valueOf(x.charAt(i));
14.         deque.add(new Document(
15.                     event.getOffset()+i,in));
16.     }
17.     for(int j = move;j>=0;j--){
18.         Document tempDoc = new Document(
19.                     temp.get(j).getOffset()+1,
20.                     temp.get(j).getText());
21.         deque.add(tempDoc);
22.     }
23. }else {
24.     String x = event.getText();
25.     for(int i = 0;i<x.length();i++){
26.         String in = String.valueOf(x.charAt(i));
27.         deque.add(new Document(
28.                     event.getOffset()+i,in));
29.     }
30. }
```

Kode Sumber 4.9 Kode untuk menghandle insertion

Isi dari *dequeue* akan dijadikan *input* untuk klasifikasi. Ketika di dalam *dequeue* terdapat titik koma (“ ;” / *titik koma*) atau kurung kurawal tutup (“}”), maka semua isi dari *dequeue* di-*pop* dan dijadikan sebuah *string* menggunakan *String Builder*. Karena hal ini terjadi ketika *user* sudah mengetikkan salah satu dari kedua *input* di atas maka *dequeue* akan kosong. Jika *user* melakukan

perubahan (*deletion*) maka *plug-in* akan mengeluarkan *nullpointer exception*.

```
1.  if(deque2.peekLast().getText().contains(";")||deque2.
    peekLast().getText().contains("}")){
2.      // detecting end of statement
3.      StringBuilder builder = new StringBuilder();
4.      while(deque2.size()>0){
5.          String str = deque2.pop().getText();
6.          builder.append(str);
7.      }
8.      //              System.out.println("Deque2 "+buil
    der.toString());
9.      myFile.classifying(builder.toString());
10. }
```

Kode Sumber 4.10 Kode untuk mengeluarkan isi dequeue

4.2.3 Implementasi StateMachine

Statemachine yang digunakan dalam tugas akhir ini merupakan hasil *generate* dari *file* xml menggunakan *library* stateforge-javabuilder. Dalam *file* xml dari *statemachine* ini terdapat beberapa atribut seperti *events*, *state*, *transition* dan *settings*.

```
1.  <sm:StateMachine xmlns:sm="http://www.stateforge.com/
    StateMachineJava-
    v1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
    instance" xsi:schemaLocation="http://www.stateforge.c
    om/StateMachineJavav1 http://www.stateforge.com/xsd/S
    tateMachineJava-v1.xsd">
2.      <!-- General settings -->
3.      <settings asynchronous="false"
4.
    nameSpace="com.jaeger.tugasakhir.Classification">
5.          <object instance="action" class="Classificati
    onAction"/>
6.      </settings>
```

Kode Sumber 4.11 *File* fsmjava untuk spesifikasi *state machine*

Dalam *settings* dispesifikasikan atribut *class* yang terbentuk nantinya. *Event* memiliki *id* masing-masing. Fungsi dari *event* adalah sebagai *trigger* untuk berpindah antara *state* satu dengan *state* lainnya.

```
<events>
    <eventSource name="Event">
        <event id="evKeyword"/>
        <event id="evClass"/>
        <event id="evUnknown"/>
        <event id="evBrackets"/>
        <event id="evCurlyBrackets"/>
        <event id="evSemicolon"/>
        <event id="evExtends"/>
        <event id="evImplements"/>
        <event id="evEquals"/>
        <event id="evNew"/>
    </eventSource>
</events>
```

Kode Sumber 4.12 Events yang ada di state machine

```
<state name="Classification">
  <state name="Initial_State">
    <transition event="evKeyword"
                nextState="Keyword"/>
    <transition event="evClass"
                nextState="Class_State"/>
    <transition event="evExtends"
                nextState="Extends_State"/>
    <transition event="evImplements"
                nextState="Implements_State"/>
    <transition event="evUnknown"
               nextState="Name_Identifier_State"/>
  </state>
  <state name="Class_State">
        <transition event="evUnknown"
              nextState="Name_Identifier_State"/>
  </state>
```

Kode Sumber 4.13 State dari state machine bagian 1

State adalah definisi dari *state* apa saja yang akan di-*generate* dalam *statemachinebuilder*. *Transition* mendefinisikan sebuah perubahan dalam *statemachine*. *Transition* didefinisikan oleh *id event*, kondisi dari *event* dan beberapa set aksi untuk dilakukan misalnya *state* selanjutnya yang dituju.

```
1.  <state name="Keyword_State">
2.     <transition event="evClass"
3.                 nextState="Class_State"/>
4.     <transition event="evUnknown"
5.                 nextState="Name_Identifier_State"/>
6.     <transition event="evKeyword"
7.                 nextState="Keyword_State"/>
8.  </state>
9.  <state name="Name_Identifier_State">
10.     <transition event="evBrackets"
11.                 nextState="Bracket_State"/>
12.     <transition event="evCurlyBrackets"
13.                 nextState="Final_State"/>
14.     <transition event="evSemicolon"
15.                 nextState="Final_State"/>
16.     <transition event="evExtends"
17.                 nextState="Extends_State"/>
18.     <transition event="evImplements"
19.                 nextState="Implements_State"/>
20.     <transition event="evEquals"
21.                 nextState="Equal_State"/>
22. </state>
23. <state name="Equal_State">
24.      <transition event="evUnknown"
25.                 nextState="Name_Identifier_State"/>
26.      <transition event="evNew"
27.                 nextState="New_State"/>
28. </state>
29. <state name="Extends_State">
30.       <transition event="evUnknown"
31.                 nextState="Name_Identifier_State"/>
32. </state>
33.
```

Kode Sumber 4.14 State dari state machine bagian 2

```
<state name="Implements_State">
    <transition event="evUnknown"
                nextState="Name_Identifier_State"/>
</state>
<state name="New_State">
     <transition event="evUnknown"
                nextState="Name_Identifier_State"/>
</state>
<state name="Bracket_State">
     <transition event="evUnknown"
                 nextState="Bracket_State"/>
     <transition event="evKeyword"
                 nextState="Bracket_State"/>
     <transition event="evCurlyBrackets"
                 nextState="Final_State"/>
     <transition event="evSemicolon"
                 nextState="Final_State"/>
</state>
<state name="Final_State" kind="final">
</state>
</state>
```

Kode Sumber 4.15 State dari state machine bagian 3

Hasil *generate* dari *file statemachine* xml tersebut adalah *class-class* dari *state* yang didefinisikan salah satu contohnya adalah *class* statemachineClass_Statestate pada Kode Sumber 4.16 sampai Kode Sumber 4.17.

```
1.  public class statemachineClass_StateState
2.      extends statemachineClassificationState
3.  {
4.      private final static statemachineClass_StateStat
    e instance =
5.              new statemachineClass_StateState();
6.      /**
7.       * Protected Constructor
8.       */
9.      protected statemachineClass_StateState() {
10.         setName("Class_State");
11.         setStateParent(statemachineClassificationSta
    te.getInstance());
12.     }
13.     /**
14.      * Get the State Instance
15.      */
16.     public static statemachineClassificationState ge
    tInstance() {
17.         return instance;
18.     }
19.
```

Kode Sumber 4.16 Salah satu *class* hasil *generate* dari *library* StateForge

Class yang dihasilkan oleh *library* Stateforge berdasarkan spesifikasi dari *states* yang tertulis di *file* statemachine.fsmjava. Setiap *state* akan diubah menjadi sebuah *class*. *Event transisition* dalam *state* berubah menjadi *method* dalam *state*.

```
20.     @Override
21.     public void onEntry(statemachineContext context
    ) {
22.         context.getObserver().onEntry(context.getNa
    me(), this.getName());
23.         ClassificationAction action = context.getCl
    assificationAction();
24.     }
25.
26.     /**
27.      * onExit
28.      */
29.     @Override
30.     public void onExit(statemachineContext context)
     {
31.         context.getObserver().onExit(context.getNam
    e(), this.getName());
32.     }
33.     /**
34.      * Event id: evUnknown
35.      */
36.     public void evUnknown(statemachineContext conte
    xt) {
37.         ClassificationAction action = context.getCl
    assificationAction();
38.         // Transition from Class_State to Name_Iden
    tifier_State triggered by evUnknown
39.         // The next state is within the context sta
    temachineContext
40.         context.setTransitionName("evUnknown");
41.         com.stateforge.statemachine.algorithm.State
    Operation.processTransitionBegin(context, statemach
    ineName_Identifier_StateState.getInstance());
42.         com.stateforge.statemachine.algorithm.State
    Operation.processTransitionEnd(context, statemachin
    eName_Identifier_StateState.getInstance());
43.         return ;
44.     }
45. }
```

Kode Sumber 4.17 Salah satu class hasil *generate* dari *library* StateForge bagian 2

4.2.4 Implementasi Klasifikasi

String yang dihasilkan oleh *dequeue* diklasifikasikan menggunakan *statemachine* hasil *generate* oleh *library* StateForge.

Untuk mengimplementasikan proses klasifikasi, dibuat sebuah *class* WriteFile. *Class* ini memiliki *method* writeXMLActivities, writeXMLRaw, classifiying. *Method* writeXMLActivities dan writeXMLRaw menggunakan DOM Parser untuk membuat *file* dengan tipe *file* xml. Sedangkan *method* classifiying digunakan untuk memecah *string* dari *dequeue* menjadi *string* per kata. Di sini juga dideteksi apakah *file* yang masuk merupakan *block statement* atau bukan. Jika bukan, maka di dalam *method* akan dibuat sebuah *object Classification* yang merupakan representasi dari sebuah *statemachine*.

```
1.  public void classifying(String string) {
2.      this.sequenceNum++;
3.      String classified = null;
4.      Classification main = new Classification();
5.      String keyword[] = {
6.      "abstract","continue","for","new","switch"
7.      ,"assert","default","package","synchronized"
8.      ,"boolean","do","if","private"
9.      ,"break","double","implements","protected","throw"
10.     ,"byte","else", "import","public","throws"
11.     ,"case","enum", "instanceof","return","transient"
12.     ,"catch","extends","int","short","try"
13.     ,"char","final","interface","static","void"
14.     ,"class","finally","long","strictfp","volatile"
15.     ,"const","float","native","super","while","String"}
    ;
16.     List<String> keywords = Arrays.asList(keyword);
```

Kode Sumber 4.18 *Method* classifying

Lebih detail pada *method* classifying ini terdapat *counter* untuk *sequencenumber*, dan *list* dari *keyword* yang ada pada pemrograman Java. Parameter dari *method* ini berupa *string* dari

dequeue. Dalam *method* ini juga terdapat *variable classified* untuk menyimpan kode dari hasil klasifikasi

Kemudian dalam *string* akan dipecah berdasarkan *whiteSpace*. Dalam perulangan setiap string akan dicek apakah merupakan *keyword* untuk *block statement*. Jika iya maka akan dikeluarkan dari perulangan dan set *variable classified* menjadi kode untuk *block statement*. Proses ini ditunjukkan oleh Kode Sumber 4.19. Jika tidak masuk ke dalam *block statement* maka *string* akan diklasifikasikan menggunakan *state machine*. Pada setiap perulangan, jika *string* termasuk ke dalam *token* tertentu maka akan men-*trigger event* yang sesuai dengan *token* tersebut dengan begitu *state machine* akan berpindah *state* sesuai dengan *transition design* yang sudah dispesifikasikan dalam *file* xml nya.

```
1.  String testSplit[] = string.split(" ");
2.     for(String currString: testSplit){
3.
4.        if(currString.contains("if(")
5.           || currString.equals("if")){
6.                    // if block
7.                  classified = "if";
8.        }else if(currString.equals("switch")
9.           ||currString.contains("switch(")){
10.                  //switch block
11.                  classified = "switch";
12.       }else if(currString.equals("for")
13.          ||currString.contains("for(")){
14.                   // for block
15.                   classified = "for";
16.       }else if(currString.equals("try")
17.          || currString.contains("try{")){
18.                   // try block
19.                   classified = "try";
20.       }else if(currString.contains("case")
21.          || currString.equals("case")){
22.                   // case block
23.                   classified = "case";
24.       }else if(currString.contains("else{")|| currString.
    equals("else")){
25.                   // else block
26.                   classified = "else";
27.        }else if(currString.equals("break;")){
28.                   //break block
29.                   classified = "break";
30.         }else if(currString.equals("return")){
31.                   classified = "return";
32.         }
```

Kode Sumber 4.19 Kode untuk mengatur block statement

```
1.  else{
2.      currString = currString.replaceAll("\\s+", "");
3.      if(keywords.contains(currString)){
4.          main.evKeyword();
5.          if(currString.equals("class")){
6.              main.evClass();
7.          }
8.          if(currString.equals("new")){
9.              main.evNew();
10.             main.state();
11.         }
12.         if(currString.equals("extends")){
13.             main.evExtends();
14.             main.state();
15.         }
16.         if(currString.equals("implements")){
17.             main.evImplements();
18.             main.state();
19.         }
20. }
21. else if(currString.contains("=")){
22.         main.evEquals();
23.     }else{
24.         main.evUnknown();
25.         main.state();
26.         if(currString.contains("(")){
27.             main.evBrackets();
28.             main.state();
29.         }
30.         if(currString.contains(";")){
31.             main.evSemicolon();
32.         }
33.         if(currString.contains("{")){
34.             main.evCurlyBrackets();
35.         }
36.     }
37. }
```

Kode Sumber 4.20 Kode untuk mengatur statement

Di setiap akhir perulangan terdapat pengecekan *state* yang aktif dari *state machine* sehingga didapatkan *state* apa saja yang telah dilalui oleh *state machine* tersebut. Jika sudah mencapai Final_State maka kode yang dikeluarkan adalah kode berdasarkan *state* yang telah dilalui.

```
1.  if(classified!=null){
2.      try {
3.          this.writeXMLClassified(classified, string);
4.      } catch (IOException e) {
5.          e.printStackTrace();
6.      }
7.  }
8.
9.  if(main.getCurrentState().equals("Final_State")){
10.     String word = main.calculation();
11.     try {
12.         this.writeXMLClassified(word, string);
13.     } catch (IOException e) {
14.         e.printStackTrace();
15.     }
16. }
```

Kode Sumber 4.21 Kode untuk melakukan pengecekan statemachine

Dalam menulis ke dalam *file* XML terdapat dua tipe xml yang dibedakan secara struktur. *Method* writeXMLRaw untuk menulis filename-raw.xml dan writeXMLActivities digunakan untuk menulis filename-activities.xml. Dalam *activities* ini ditambahkan kode yang sudah dihasilkan saat klasifikasi sebelumnya. Hasil dari *log* aktivitas ini dapat ditemukan di dalam direktori yang sama dengan *file* Java yang dituliskan.

```
1.  public void writeXMLActivities
    (String typeContent,String textContent) throws IOException
    {
2.      try {
3.          DocumentBuilderFactory docFactory =
4.              DocumentBuilderFactory.newInstance();
5.          DocumentBuilder docBuilder =
6.              docFactory.newDocumentBuilder();
7.          Document doc =
8.              docBuilder.parse(this.pathClassified);
9.          Node activities = doc.getFirstChild();
10.         Element activity = doc.createElement("activity");
11.         Attr attrType = doc.createAttribute("type");
12.         Attr attrSequence =
13.             doc.createAttribute("sequence");
14.         attrSequence.setValue(
15.             String.valueOf(this.getSequence()));
16.         attrType.setValue(typeContent);
17.         activity.setAttributeNode(attrSequence);
18.         activity.setAttributeNode(attrType);
19.         Element text = doc.createElement("text");
20.         text.appendChild(
21.              doc.createTextNode(textContentInside));
22.         activity.appendChild(text);
23.         activities.appendChild(activity);
24.         // Adding
25.         TransformerFactory transformerFactory =
26.               TransformerFactory.newInstance();
27.         Transformer transformer =
28.               transformerFactory.newTransformer();
29.         DOMSource source = new DOMSource(doc);
30.         StreamResult result = null;
31.         result = new StreamResult(
32.               new File(this.pathClassified));
33.         transformer.transform(source, result);
34.     }
35. }
```

Kode Sumber 4.22 Kode untuk menuliskan Activities.xml

```
1.  public void writeXMLRaw(DocumentEvent event) throws IOExc
    eption{
2.      try {
3.          DocumentBuilderFactory docFactory =
4.               DocumentBuilderFactory.newInstance();
5.          DocumentBuilder docBuilder =
6.               docFactory.newDocumentBuilder();
7.          Document doc = docBuilder.parse(this.pathRaw);
8.          Node activities = doc.getFirstChild();
9.          Element raw = doc.createElement("raw");
10.         Attr attrSequence =
11.              doc.createAttribute("sequence");
12.         attrSequence.setValue(
13.               String.valueOf(this.getSequence()));
14.         Attr attrOffset = doc.createAttribute("offset");
15.         attrOffset.setValue(
16.               String.valueOf(event.getOffset()));
17.         Attr attrLength = doc.createAttribute("length");
18.         attrLength.setValue(
19.               String.valueOf(event.getLength()));
20.         Element text = doc.createElement("text");
21.         text.appendChild(
22.                doc.createTextNode(event.getText()));
23.         raw.appendChild(text);
24.         activities.appendChild(raw);
25.         TransformerFactory transformerFactory =
26.                TransformerFactory.newInstance();
27.         Transformer transformer =
28.                transformerFactory.newTransformer();
29.         DOMSource source = new DOMSource(doc);
30.         StreamResult result = null;
31.         result = new StreamResult(
32.                new File(this.pathRaw));
33.         transformer.transform(source, result);
34.     }
35. }
```

Kode Sumber 4.23 Kode untuk menuliskan raw.xml

(Halaman ini sengaja dikosongkan)

BAB 5
HASIL UJI COBA DAN EVALUASI

Bab ini berisi penjelasan mengenai skenario uji coba dan evaluasi pada *plug-in* yang ditambahkan pada IDE Eclipse. Hasil uji coba didapatkan dari implementasi BAB 4 dengan dua skenario yang berbeda. Bab ini berisikan lingkungan pengujian, data pengujian dan hasil pengujian.

5.1 Lingkungan Pengujian

Lingkungan pengujian pada uji coba *plug-in* Eclipse ini adalah perangkat keras dan perangkat lunak yang spesifikasinya dapat dilihat pada Tabel 5.1.

Tabel 5.1 Lingkungan pengujian

Perangkat	Jenis Perangkat	Spesifikasi
Perangkat Keras	Prosesor	Intel(R) Core(TM) i5-2.6 GHz
	Memori	4 GB
Perangkat Lunak	Sistem Operasi	Windows 10
	Perangkat Pengembang	Eclipse Neon

5.2 Skenario Pengujian

Untuk menguji *plug-in* ini maka dibuat dua skenario. Dua skenario tersebut dibuat untuk mengakomodasi seluruh aktivitas programming yang disebutkan dalam tugas akhir ini. Dalam skenario pertama yaitu *game* Tic Tac Toe, tidak terdapat *inheritance* seperti *extends* dan *implement*. Aktivitas programming itu didapatkan dari skenario kedua yaitu *game* Tamagotchi. Pola pengujian adalah skenario pertama diujikan pada tiga orang subjek. satu orang *beginner programmer*, satu *intermediate programmer*

dan satu *expert programmer*. Untuk skenario kedua diujikan pada dua orang *intermediate programmer*.

Skenario 1

Programmer diminta untuk membuat sebuah *game* Tic Tac Toe. Penulis sudah menyiapkan struktur dari *game* Tic Tac Toe, *programmer* dapat menggunakannya untuk membantu saat menuliskan kode di IDE Eclipse yang sudah terdapat *plug-in* didalamnya. Struktur *game* yang diminta untuk dibuat adalah sebagai berikut:

- *Case* 1: *Programmer* Java dengan kemampuan *intermediate* diberikan struktur program secara detail

Buatlah sebuah *game* Tic Tac Toe, dokumen ini hanya menyediakan stuktur dari program yang harus ada di program yang dibuat. Tidak perlu menuliskan kode sesuai dengan urutan yang ada di dokumen ini.

Enum GameState

- *PLAYING, DRAW, CROSS_WON, NOUGHT_WON*

Enum Seed

- *EMPTY, CROSS , NOUGHT*

Class Cell

- variabel content dengan tipe data Seed
- variabel row, col dengan tipe data int
- *Contructor class* cell dengan parameter int row , int col
    - Inisialisasi *class* variabel row dengan row *input*
    - Inisialisasi *class* col dengan col *input*
    - Panggil *method clear*
- *Method* clear
    - Isi variabel *content* dengan Seed.*EMPTY*
- *Method* paint
    - Kondisi untuk variabel *content*
    - Jika isi dari *content* adalah *CROSS* maka *print* (“ X ”);

- Jika isi dari *content* adalah *NOUGTH* maka *print* (“ 0 ”);
- Jika isi dari *content* adalah *Empty* maka *print* (“ ”);

Class Board

- variabel dengan *modifier public static final* dengan tipe data *int* bernama ROWS = 3
- variabel dengan *modifier public static final* dengan tipe data *int* bernama COLS = 3
- Array 2 dimensi dari Cell bernama cells
- variabel *currentRow* dan *currentCol* dengan tipe data *int*
- *Constructor* Board
    - Mengisi variabel cells dengan menginstansiasi *object* Cell dengan parameter ROWS dan COLS
    - Untuk setiap ROWS
        - Untuk setiap COLS
            - Isi variabel cells ke-ROWS ke COLS dengan object Cell ke row dan ke col
- *Method init* dengan *return* type void
    - Untuk setiap kolom dan baris dari cell panggil *method* clear
- Method isDraw dengan *return type* boolean
    - Cari apakah ada cell yang kosong jika ada
        - kembalikan *value false*
        - Jika tidak kembalikan *value true*
- Method hasWon dengan parameter *type* Seed nama theSeed dan return *value* Boolean
    - Cek apakah dalam 1 diagonal atau satu horizontal atau satu vertical, variabel cells.content memiliki isi yang sama
- *Method* Paint dengan *return value* void
    - Untuk setiap ROWS
        - Untuk setiap COLS

- Panggil *method* paint di cells ke row dan ke col
- Jika col kurang dari COLS -1 maka print (“ | “)

▪ *Print* enter
▪ Jika row < ROWS-1 maka print (“ ---------- “)

Class Game Main
- variabel board dengan tipe data Board
- variabel currentState dengan tipe data GameState
- variabel currentPlayer dengan tipe data Seed
- variabel in dengan *modifier static* tipe data Scanner dengan *value* Object Scanner dengan parameter System.in
- *Constructor* GameMain
    - Isi variabel board dengan Object Board
    - Panggil *method* initGame
    - Do block statement
        - Panggil *method* playerMove dengan parameter currentPlayer
        - Panggil *method* paint dari variable board
        - Panggil *method* updateGame dengan parameter currentPlayer
        - Cek currentState apakah sama dengan salah satu dari GameState jika ya *print* yang pihak yang menang
        - Isi *variable* currentPlayer dengan player lawan yang bermain saat ini
    - While *statement* currentState adalah GameState.Playing
- *Method* initGame
    - Panggil *method* init dari variabel board;
    - Isi variabel currentPlayer dengan Seed.CROSS

        - Isi variabel currentState dengan GameState.Playing
- *Method* playerMove dengan parameter tipe Seed variabel theSeed *return value* void
    - variabel validInput tipe Boolean isi dengan value false
    - Do block *statement*
        - Jika theSeed adalah CROSS maka *print* (“X move first row[0-3] col[0-3]”)
        - Jika yang lain *print* (“O move first row[0-3] col[0-3]”)
        - variabel tipe int nama row isi dengan *value* in.nextInt() – 1
        - variabel tipe int nama col isi dengan *value* in.nextInt() – 1
        - Cek apakah nilai yang dimasukkan benar dan apakah cells pada row dan kolom yang dimasukkan content-nya adalah Seed.EMPTY
            - Isi variabel board.cells[row][col].content dengan theSeed
            - Isi variabel currentRow dari *variable* board dengan row
            - Isi variabel currentCol dari *variable* board dengan col
            - Isi variabel inputValid dengan True
        - Else
            - *Print* (“ Input tidak valid masukkan lagi ”)
    - While block *statement* parameter !validInput
- *Method* updateGame parameter tipe Seed nama theSeed dengan *return type* void
    - Cek apakah sudah ada yang menang dengan memanggil board.hasWon(theSeed)
        - Isi variabel currentState dengan variabel yang sudah memenangkan game
    - Cek apakah draw
        - Isi variabel currentState dengan GameState.DRAW
- Method main java

- Panggil *constructor* dari GameMain

- Case 2: *Programmer* Java yang berpengalaman hanya dengan spesifikasi kebutuhan

Buat *game* Tic Tac Toe dengan spesifikasi sebagai berikut:

- Papan permainan dengan ukuran 3 x 3
- *Input* dan *output* berupa *console*
- Harus *object oriented*
- Player terdiri dari CROSS dan NOUGHT
- Player secara bergantian bermain dengan CROSS bermain terlebih dahulu

Skenario Test Kedua

Membuat sebuah *text-based gaming* Tamagotchi *battle*. Di mana pemain bisa memiliki sebuah *pet* (tidak perlu *multiplayer* dan tidak perlu AI). Silahkan buat 3 jenis pet. Konsep *object oriented* yang harus diimplementasikan *implementation, inheritance*.

5.3 Hasil Pengujian

Pengujian dilakukan pada 5 orang *programmer* Java. Pada semua kasus pengujian, *plug-in* tidak langsung berjalan saat *workbench* Eclipse sudah siap. Setelah dilakukan analisis, ternyata *plug-in* membutuhkan *user* untuk mengaktifkan *editor* pada dokumen yang ingin di-*edit*. Hasil dari pengujian dipaparkan berdasarkan hasil per orang beserta dengan evaluasi program.

Hasil uji *beginner programmer* skenario 1

Pada pengujian pertama dengan *plug-in* yang terpasang pada IDE Eclipse, dilakukan dalam durasi 1 jam. Subyek uji coba terlihat sangat tertekan karena tidak terbiasa menuliskan program java sehingga hanya terbentuk tiga *class* yang diimplementasikan dari lima *class* yang ada, yaitu GameState.java, Cell.java, dan

Seed.java. Semua *class* dibentuk secara manual tanpa ada bantuan dari IDE. Hasil dari GameState-activities.xml sebagai berikut

```
<activity sequence="1" type="">
    <text>ublic enum GameState {

            }
    </text>
</activity>
```

Gambar 5.1 Hasil *output plug-in beginner programmer* bagian 1

Activity tersebut terlihat aneh karena *text* yang tertangkap hanya ublic padahal dalam file asli *text* ditulis *public*. Hal ini terjadi karena pada saat menuliskan *code* subjek melakukan kesalahan ketik dengan menuliskan titik koma(“;”) setelah p sehingga *dequeue* di-*pop* semua isinya mengakibatkan *dequeue* tidak dapat melakukan *capture* yang benar meskipun terdapat *handler* untuk *deletion*.

Pada Seed-activities.xml yang didapatkan dari Seed.class. didapatkan hasil seperti berikut

```
<activity sequence="2" type="">
    <text>

        PLAYING, DRAW, CROSS_WON, NOUGHT_WON

        public enum GameState {

        PLAYING, DRAW, CROSS_WON, NOUGHT_WON

        }
    </text>
</activity>
```

Gambar 5.2 Hasil *output plug-in beginner programmer* bagian 2

Terbentuknya *activity* di atas karena subjek melakukan *copy-paste* program dari kode GameState.java dan kemudian mengubah isi dan nama-namanya saja. Ketika subyek melakukan *copy-paste* program sudah mendeteksi *pop-trigger* yaitu kurung kurawal tutup (“}”). Sehingga saat subyek mengganti nama-namanya maka tidak akan muncul dalam hasil *output plug-in*.

Untuk Cell-activities.xml dihasilkan *output* Gambar 5.3.

```
1.  <activity sequence="3" type="AV">
2.      <text>SeeeSeed content;
3.
4.          mpint row, col;
5.          typublic Cell, crross, nought(){
6.          }
7.      </text>
8.  </activity>
9.  <activity sequence="4" type="IV">
10.     <text>int row, int colthis.row = row;</text>
11. </activity>
12. <activity sequence="5" type="IV">
13.     <text>
14.
15.         this.col = col;
16.     </text>
17. </activity>
18. <activity sequence="6" type="CM">
19.     <text>
20.
21.         clear();
22.     </text>
23. </activity>
24.
```

Gambar 5.3 Cell-activities.xml

```
<activity sequence="7" type="AM">
    <text>

        private void clear(){



        }
    </text>
</activity>
<activity sequence="8" type="IV">
    <text>content = Seed.empty;</text>
</activity>
```

Gambar 5.4 Cell-activities.xml

Dapat dilihat masih banyak *bug* seperti pada *activity sequence* 3 di mana program tidak berhasil melakukan *end statement* dengan baik yang kemudian diselesaikan pada program versi selanjutnya. Namun jika masukan benar maka klasifikasi yang dihasilkan juga benar seperti pada *activity* dengan *sequence* nomor 5, 6, 7, 8.

Dari uji-coba pertama ini didapatkan bahwa masih banyak *bug* dalam hal menangkap *input* secara *general* dari *keystroke* subyek. Untuk klasifikasi sudah teruji benar.

Hasil uji coba *intermediate programmer* skenario 2

Pada uji coba ini subjek sering menggunakan *tools* dari Eclipse IDE seperti *generate getter* dan *setter* dan *autocomplete*. Subjek juga sering melakukan *copy-paste* dalam menuliskan *code*. Subjek ini menyelesaikan tes dalam waktu 3 jam, tapi dengan selang waktu dalam menuliskan kode. Secara keseluruhan, program dengan baik melakukan klasifikasi. Namun terdapat beberapa *bug* yang muncul.

Dari sebagian hasil di atas, didapatkan bahwa *autogenerate* dari IDE dapat dihasilkan *klasifikasi* yang tepat.

```
<activity sequence="13" type="AM">
    <text>
        public void setName(String name) {
        this.name = name;
        }
    </text>
</activity>
<activity sequence="14" type="AM">
    <text>
        public String getName() {
        return name;
        }
    </text>
</activity>
```

Gambar 5.5 Hasil *output intermediate programmer* skenario 2 bagian 1

Namun terdapat beberapa kesalahan dalam *input* ke dalam *dequeue* seperti pada *activity sequence* 15 yang terjadi karena isi dari *dequeue* terdapat isi dari *aktivitas* yang sebelumnya. Jadi ketika melakukan *autocomplete* IDE membuat kode sampai ke *end statement* yaitu titik koma sehingga langsung di-*pop* oleh *dequeue*. Ketika subjek melakukan perubahan isi pada hasil *autocomplete* tersebut subjek tidak perlu membuat *end statement* sehingga isi *dequeue* tidak dikeluarkan dan melanjutkan ke aktivitas selanjutnya.

```
15. <activity sequence="15" type="CM">
16.     <text>
17.         s eating" System.out.println(this.name+
    " attack "+enemy.getName());
18.     </text>
19. </activity>
20. <activity sequence="17" type="AM">
21.     <text>
22.         public Monster(String name, String type
    , Integer health) {
23.         super();
24.         this.name = name;
25.         this.type = type;
26.         this.health = health;
27.         }
28.     </text>
29. </activity>
```

Gambar 5.6 Hasil *output intermediate programmer* skenario 2 bagian 2

Untuk *activity sequence* 16 diklasifikasikan sebagai “AM” karena ketika *string* hasil *copy-paste* masuk klasifikasi dan diproses per kata, *state machine* sudah mencapai Final_STATE ketika *Add Method* terdeteksi sehingga isi dari *method* tersebut tidak terklasifikasikan.

Hasil uji coba *intermediate programmer* pada skenario 1 case 1

Pada uji coba ketiga ini digunakan program versi kedua yang dihasilkan dari evaluasi uji coba pertama dan kedua. Dilakukan dalam dua kali sesi setiap sesi memiliki durasi waktu 1 jam.

Saat subjek membuat GameState.java dan juga Seed.java, subjek menggunakan *file create enum* dari IDE sehingga tidak tercatat pembuatan *enum* oleh klasifikasi. Dalam uji coba kali ini juga sering terdapat *block statement* seperti if dan for. Hasil dari klasifikasi yang didapatkan memiliki tingkat kebenaran yang baik tetapi masih terdapat kesalahan ketika konteks dari *statement* yang ditulis tidak sesuai sebagai contohnya seperti pada Gambar 5.7.

```
1.  <activity sequence="1" type="IV">
2.      <text>
    cell = Cell[this.ROWS][this.CCOLS;</text>
3.  </activity>
4.  <activity sequence="2" type="IO">
5.      <text>
6.
7.          Cell cells = new Cell(ROWS, COLS);
8.      </text>
9.  </activity>
10. <activity sequence="3" type="IO">
11.     <text>
12.
13.         this.cells = new new Cell[ROWS][COLS];
14.     </text>
15. </activity>
16. <activity sequence="4" type="for">
17.     <text>for (Cell[] cells2 : cells) {
18.         }
19.     </text>
20. </activity>
21. <activity sequence="5" type="for">
22.     <text>
23.
24.         for(int row =0; row<this.ROWS; row++ ){
25.         }
26.     </text>
27. </activity>
```

Gambar 5.7 Hasil *output intermediate programmer* dengan skenario 1 bagian 1

Pada *activity* dengan *sequence* 1 terdapat kesalahan dalam *middle insertion* sehingga terbentuk *string* yang salah dari *dequeue*. Namun hal tersebut tidak mempengaruhi *dequeue* karena *state machine* menerima *event* dengan benar. Dapat dilihat bahwa *for block statement* dideteksi dengan baik.

```
1.  <activity sequence="11" type="CM">
2.      <text>playerMove(this.ccurrentPlayer;</text>
3.  </activity>
4.  <activity sequence="12" type="CM">
5.      <text>
6.          this.board.Paint();
7.      </text>
8.  </activity>
9.  <activity sequence="13" type="CM">
10.     <text>
11.         updateGame(this.ccurrentPlayer;
12.     </text>
13. </activity>
14. <activity sequence="14" type="if">
15.     <text>
16.
17.         if(this.ccurrentStat == GameState.eCROSS_W
    ON ||
18.         this.currentState == GameState.NOUGHT_WON{

19.         }
20.     </text>
21. </activity>
22. <activity sequence="15" type="">
23.     <text>.CROSS_WON{
24.         }
25.     </text>
26. </activity>
```

Gambar 5.8 Hasil *output intermediate programmer* skenario 1 bagian 2

Dari sebagian hasil di atas terdapat *activity* yang terklasifikasi dengan benar tetapi isi dari *string* salah yaitu *activity sequence* 11,13,14 hal ini terjadi karena kesalahan dalam operasi *middle insertion*. Untuk *activity sequence* 15 tidak terklasifikasi karena string hasil dari *dequeue* tidak diketahui oleh *state machine*.

```
1.  <activity sequence="43" type="CM">
2.      <text>System.out.println(x);</text>
3.  </activity>
4.  <activity sequence="44" type="AV">
5.      <text>"Input tidak valid masukan lagi"whil;</t
    ext>
6.  </activity>
7.  <activity sequence="45" type="CM">
8.      <text>e(!validInput);</text>
9.  </activity>
```

Gambar 5.9 Hasil *output intermediate programmer* skenario 1 bagian 3

Gambar 5.9 adalah hasil dari *activities* yang tidak benar karena kesalahan ketik titik koma sehingga pada *activity* 44 seharusnya while(); subjek salah menuliskan whil; sehingga langsung di-*pop* keluar dari *dequeue*. Begitu juga dengan *activity* 43 di mana dalam *autocomplete* langsung terdapat titik koma sehingga ketika subjek melakukan perubahan kode pada *statement* tersebut tidak direkam oleh *dequeue*.

Hasil uji coba *expert programmer* dengan skenario 1 case 1

Pengujian dilakukan dalam satu sesi dengan durasi waktu 43 menit. *Plug-in* yang digunakan merupakan revisi dari *plug-in* dari uji coba ketiga. *Plug-in* dengan baik menjalankan klasifikasi dan juga mencatat *sequence* dari penulisan program melalui banyak dokumen sekaligus secara berurutan.

```
<activity sequence="12" type="CM">
    <text>System.out.print(arg0);</text>
</activity>
<activity sequence="13" type="if">
    <text>X"
        if(content == Seed.NOUGHT) {
        }
    </text>
</activity>
<activity sequence="14" type="CM">
    <text>System.out.print(s);</text>
</activity>
<activity sequence="15" type="AV">
    <text>");</text>
</activity>
<activity sequence="16" type="if">
    <text>
        if(content == Seed.EMPTY {
        }
    </text>
</activity>
<activity sequence="17" type="CM">
    <text>System.out.print(s);</text>
</activity>
<activity sequence="35" type="AM">
    <text>
        Boolean isEmpty() {
        }
    </text>
</activity>
<activity sequence="36" type="if">
    <text>if(content == Seed.EMPTY) {
        }
    </text>
</activity>
<activity sequence="37" type="return">
    <text>return true;</text>
</activity>
```

Gambar 5.10 Hasil *output expert programmer* skenario 1 bagian 1

```
<activity sequence="32" type="AM">
    <text>
        Boolean isDraw() {
        }
    </text>
</activity>
<activity sequence="33" type="for">
    <text>for(int i = 0; i < ROWS; i++) {
        }
    </text>
</activity>
<activity sequence="34" type="for">
    <text>for(int j = 0; j <= COLS; j++) {
        }
    </text>
</activity>
<activity sequence="40" type="AM">
    <text>].isEmpty() == true {
        }
    </text>
</activity>
<activity sequence="41" type="return">
    <text>return false;</text>
</activity>
<activity sequence="42" type="AV">
    <text>
        return true;
    </text>
</activity>
```

Gambar 5.11 Hasil *output expert programmer* skenario 1 bagian 2

Dalam kedua kode di atas menunjukkan bahwa subjek membuat kode isEmpty setelah dirasa *method* itu dibutuhkan kemudian kembali meneruskan kode pada Gambar 5.10.

```
<activity sequence="60" type="CM">
    <text>" | "
        System.out.print(s);
    </text>
</activity>
<activity sequence="61" type="if">
    <text>"\n"
        if(i < ROWS-
    1) System.out.print(s);</text>
</activity>
```

Gambar 5.12 Hasil *output expert programmer* skenario 1 bagian 3

Masih terdapat masalah ketika subjek melakukan *autocomplete* pada *syntax* yang langsung memberikan *end statement*/titik koma pada kode sehingga langsung keluar dari *dequeue* padahal subyek masih melakukan perubahan di *statement* tersebut.

```
<activity sequence="44" type="AM">
    <text>Seed seed//horisontal
        for
        //vertikal
        //diagonal(int i = 0; i < ROWS; i++) {
        }
    </text>
</activity>
```

Gambar 5.13 Hasil *output expert programmer* skenario 1 bagian 4

Gambar 5.13 adalah ketika *user* berpindah kursor dan menuliskan kode di tempat lain maka tetap masuk ke dalam *dequeue* sehingga menampilkan klasifikasi yang salah. *state machine* mendeteksi *name_identifier* dari Seed kemudian terdapat

kurung dan kurung kurawal sehingga masuk ke klasifikasi *add method*.

Hasil uji coba *intermediate programmer* skenario 2

Dalam uji coba orang kelima ini digunakan *plug-in* sama dengan *plug-in* uji-coba orang keempat. Uji coba dilakukan dalam waktu 30 menit, dan difokuskan untuk melakukan *polymorphisme* dan mencoba klasifikasi *extends* dan *implements*.

```
<activity sequence="1" type="AC">
    <text>
        public class Pet{
        }
    </text>
</activity>
<activity sequence="2" type="AV">
    <text>private int health;</text>
</activity>
<activity sequence="3" type="AV">
    <text>

        private int attack;
    </text>
</activity>
<activity sequence="4" type="AM">
    <text>
        public Pet(){
        }
    </text>
</activity>
<activity sequence="5" type="IV">
    <text>this.health = 100;</text>
</activity>
```

Gambar 5.14 Hasil output intermediate programmer skenario 2 bagian 1

Subjek membuat kelas secara manual. Aktivitas *add class* terekam karena tidak langsung ada pada dokumen. Berbeda dengan uji coba yang lain

Subjek kemudian membuat *class* yang meng-*extends* kelas lain. Pada *activities* ditampilkan seperti berikut

```
<activities>
    <activity sequence="15" type="AC_EC">
        <text>

            public class Cat extends Pet{


            }
        </text>
    </activity>
    <activity sequence="16" type="AM">
        <text>public Cat(){


            }
        </text>
    </activity>
    <activity sequence="17" type="CM">
        <text>System.out.println();</text>
    </activity>
</activities>
```

Gambar 5.15 Hasil *output intermediate programmer* skenario 2 bagian 2

Subjek menuliskan deklarasi kelas dengan meng-*extend* kelas Pet. *Plug-in* mengklasifikannya dengan AC_EC.

```
<activities>
    <activity sequence="18" type="II">
        <text>"Cat created"


            public class Dog extends Pet implements
 Monster{

            }
        </text>
    </activity>
    <activity sequence="19" type="AM">
        <text>// TODO Auto-generated method stub

            @Override

            public void attackAnother(Pet x) {

            System.out.println(x);
        </text>
    </activity>
</activities>
```

Gambar 5.16 Hasil *output intermediate programmer* skenario 2 bagian 3

Saat melakukan *implement class plug-in* subyek mengetikkan *class* kemudian *extends* setelah itu baru *implement class*. Namun *plug-in* kurang tepat dalam melakukan klasifikasi hanya *implement an interface* yang didapatkan.

BAB 6
KESIMPULAN DAN SARAN

6.1 Kesimpulan

Kesimpulan yang didapatkan berdasarkan pengembangan sampai dengan hasil uji coba *plug-in* Eclipse adalah sebagai berikut:

1. *Plug-in* yang dibuat mampu mendapatkan *keystroke input* dari *user* menggunakan *documentChangeListener* yang diimplementasikan pada *editor workbench* Eclipse.
2. *Plug-in* berhasil mendefinisikan aktivitas pemrograman dengan cara mengklasifikasikan *keystoke input* mengklasifikasikan dengan menggunakan *finite state machine*.
3. *Plug-in* memproses *keystroke input* dengan *double ended queue* kemudian memisahkan berdasarkan *whiteSpace*.
4. Hasil dari *queue* dimasukkan ke dalam *state machine* kemudian berdasarkan *history state* dari *state machine* dihasilkan *output* berupa *log* aktivitas.

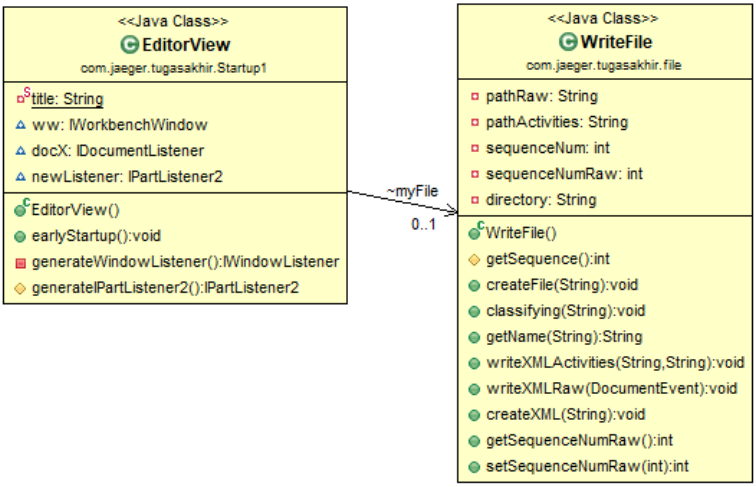
6.2 Saran

Saran yang diberikan terkait pengembangan tugas akhir ini adalah sebagai berikut:

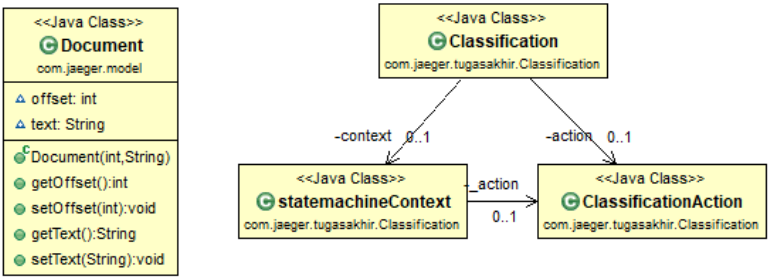
1. Membuat *plug-in* ini mampu bisa mendeteksi aktivitas pemrograman dengan mengetahui letak perubahan. Sehingga *plug-in* tahu mana kode yang masuk ke dalam *state machine.*
2. Menangkap *user input* dapat dicoba dengan cara yang lain. *Plug-in* ini menggunakan *dequeue* untuk menangani perilaku pemrograman dari *programmer*. Penggunaan struktur data lain dapat dicoba untuk pengelolaan keystroke yang lebih baik.
3. Melakukan klasifikasi yang lebih tinggi tingkatnya ketika ada *complex statement* seperti parameter dalam *method,* konteks saat menuliskan kode. Contohnya apakah *user* sedang menuliskan kode untuk variabel *class* atau hanya variabel lokal.

4. Menambah *handling* untuk *file* yang dibuat secara otomatis oleh IDE ketika *create new file* atau *autocomplete* yang terdapat *end statement*.
5. Melakukan klasifikasi yang spesifik untuk aktivitas pemrograman kompleks semisal membuat sebuah variable baru dan kemudian mengisi *variabel* tersebut. Sehingga aktivitas yang tercatat bisa *atomic*.

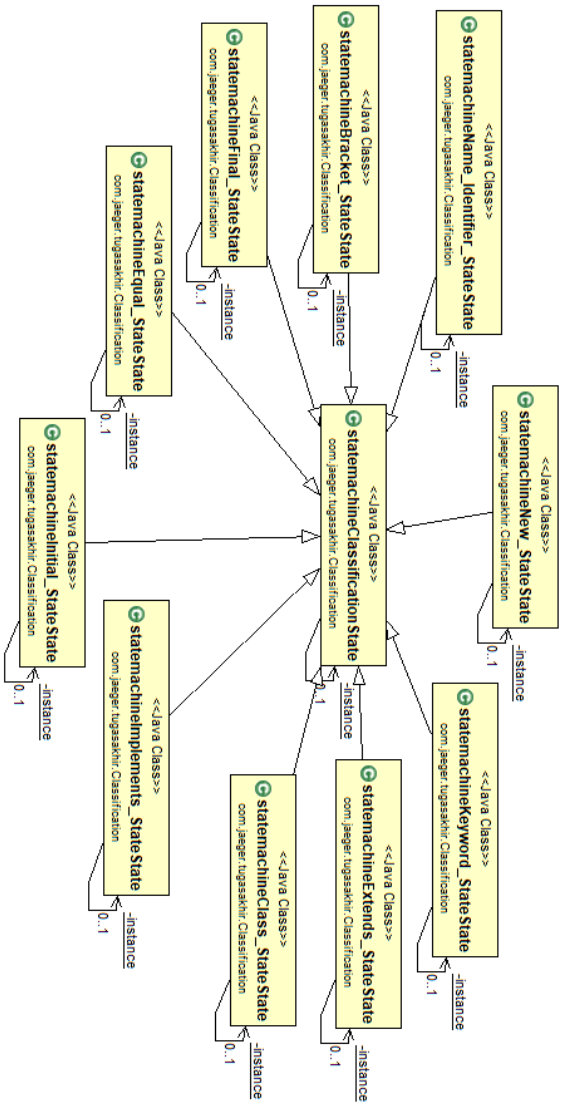
LAMPIRAN



Lampiran A Class Diagram



Lampiran B Class diagram



Lampiran C Class diagram

DAFTAR PUSTAKA

[1] T. Omori dan K. Maruyama, "A Change-aware Development Environment by Recording Editing Operations of Source Code," in *Proceedings of the 2008 International Working Conference on Mining Software Repositories*, New York, NY, USA, 2008, hal. 31–34.

[2] "Help - Eclipse Platform." [Daring]. Tersedia pada: http://help.eclipse.org/kepler/index.jsp?topic=%2Forg.eclipse.platform.doc.isv%2Freference%2Fapi%2Forg%2Feclipse%2Fjface%2Ftext%2FIDocument.html. [Diakses: 19-Feb-2017].

[3] tutorialspoint.com, "Eclipse Explore Windows," *www.tutorialspoint.com*. [Daring]. Tersedia pada: https://www.tutorialspoint.com/eclipse/eclipse_explore_windows.htm. [Diakses: 06-Jun-2017].

[4] "Notes on the Eclipse Plug-in Architecture." [Daring]. Tersedia pada: https://eclipse.org/articles/Article-Plug-in-architecture/plugin_architecture.html. [Diakses: 31-Mei-2017].

[5] "State machine tools: code generator & state diagram editor." [Daring]. Tersedia pada: http://www.stateforge.com/Help/state-machine-tools.aspx. [Diakses: 17-Apr-2017].

[6] B. G. Ryder dan F. Tip, "Change Impact Analysis for Object-oriented Programs," in *Proceedings of the 2001 ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering*, New York, NY, USA, 2001, hal. 46–53.

[7] "Java (programming language)," *Wikipedia*. 08-Apr-2017.

[8] "Java Tutorial 2 - Syntax and Grammar." [Daring]. Tersedia pada: http://www.cs.put.poznan.pl/mmasewicz/dydaktyka/po/java_tutorial/jatutor2.htm. [Diakses: 31-Mei-2017].

[9] "The Java® Language Specification." [Daring]. Tersedia pada: https://docs.oracle.com/javase/specs/jls/se8/html/index.html. [Diakses: 31-Mei-2017].

[10]“Deque (Java Platform SE 7).” [Daring]. Tersedia pada: https://docs.oracle.com/javase/7/docs/api/java/util/Deque.html. [Diakses: 24-Mei-2017].

BIODATA PENULIS

Andre Zachary Reinaldi merupakan anak dari pasangan Bapak Rubianto dan Ibu Khomsatun. Lahir di Madiun pada tanggal 10 Juli 1995. Penulis menempuh pendidikan formal dimulai dari TK ABA 3, SDN 02 Mojorejo, SMPN 1 Madiun, SMAN 2 Madiun dan S1 Teknik Informatika ITS. Bidang studi yang diambil oleh penulis pada saat berkuliah di Teknik Informatika ITS adalah Algoritma dan Pemrograman. Penulis aktif dalam organisasi seperti Himpunan Mahasiswa Teknik Computer-Informatika. Penulis juga aktif dalam berbagai kegiatan kepanitiaan yaitu SCHEMATICS 2013 sebagai ketua National Logic Competition. Penulis juga menyukai kegiatan sosial dan pecinta alam. Penulis memiliki hobi bermain dota 2, melakukan hal-hal yang berkaitan dengan programming. Penulis dapat dihubungi melalui email:andrezacharyreinaldi@gmail.com.